PYRAMID Technical Standard Guidance Version 1.0

This document sets out a generic approach to the implementation of the PYRAMID Reference Architecture. The PYRAMID Reference Architecture has not been created for any specific system. It is the user's responsibility to ensure that any article created using this document meets any required operational, functional and safety needs. The Author accepts no liabilities for any damages arising due to a failure of the user to verify the safety of any product produced using this document, nor for any damages caused by the user failing to meet any technical specification.

For further information regarding how you can exploit PYRAMID on your project, provide feedback, or have a technical query that you would like answering, please contact the PYRAMID Team using the following email address: PYRAMID@mod.gov.uk

CHANGE HISTORY

Date	Version	Description of Changes
February 2025	1.0	First Issue.

List of Effective Pages

571 pages in total

TABLE OF CONTENTS

TERMS AND ABBREVIATIONS USED IN THIS DOCUMENT

Definitions of terms, the meaning of acronyms and the meaning of abbreviations used in this document can be found in Appendix F: Glossary.

References to content defined in the PYRAMID Technical Standard, Ref. [1], are shown as (non-hyperlinked) green text.

REFERENCES

The reference numbers are consistent across the PYRAMID Technical Standard and PYRAMID Technical Standard Guidance documents. Only the subset of references used in this document are listed in the reference list.

PYRAMID Document References:

Reference	Author/Organisation, Date, Title, Document Number & Issue
[1]	Ministry of Defence, (February 2025). PYRAMID Technical Standard, PYD/TechStan/V1.0.
[3]	Ministry of Defence, (February 2025). PYRAMID Model, PYD/TechStanModel/V1.0.
[4]	Ministry of Defence, (September 2023). PYRAMID Exploiter's Pack Version 4.1, RCO_FUT_23_004.

Note: PYRAMID documentation is available under the Open Government Licence v3.0.

Other References:

Reference	Author/Organisation, Date, Title, Document Number & Issue
[5]	J. Borkey and T. Bradley, (2019) Effective Model-Based System Engineering. Springer.
[6]	Raistrick et al, (2004). Model Driven Architecture with Executable UML. Cambridge University Press.
[7]	T. Erl, (2005). Service-Oriented Architecture: Concepts, Technology and Design. Prentice Hall.
[8]	International Organization for Standardization / International Electrotechnical Commission, (2016), Reference Architecture for Service Oriented Architecture (SOA RA), ISO/IEC 18384:2016.
[9]	C. Raistrick, (2019), Land Data Model Methodology and Modelling Standard. [Online]. Available: https://landopensystems.mod.uk/cgi-bin/welcome.
[10]	United States Department of Defense, (2007). Aircraft/Store Electrical Interconnection System, MIL-STD-1760E.
[11]	Ministry of Defence, (March 2019). Design and Airworthiness Requirements for Service Aircraft, Defence Standard 00-970, Issue 21.
[12]	Commander Alan Cole et al. (2009). Sanremo Handbook on Rules of Engagement.
[13]	International Organization for Standardization, (2003-2015). Condition monitoring and diagnostics of machines, ISO 13374.
[15]	European Aviation Safety Agency, (2014). Airworthiness Security Process Specification, RTCA DO-326A.
[17]	RTCA SC-205 and EUROCAE WG-12, (2011). Software Considerations in Airborne Systems and Equipment Certification, RTCA DO-178C.
[18]	M. Page-Jones, (1988). Practical Guide to Structured System Design. Prentice Hall.

[19] Object Management Group, (2012). Service oriented architecture Modeling Language (SoaML) Specification.

[20] The Open Group, (2023). Technical Standard for the Future Airborne Capability Environment (FACE®), Edition 3.2.

FACE® and Future Airborne Capability Environment® are registered trademarks of The Open Group in the United States and other countries.

[21] RTCA, (2011). Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A, Radio Technical Commission for Aeronautics.

[22] RTCA, (2004). “Handbook for Object-Oriented Technology in Aviation (OOTiA),” Radio Technical Commission for Aeronautics.

[23] Assurance Case Working Group, (2019). “Goal Structuring Notation Community Standard (version 2),” GSN Working Group. [Online]. [Accessed 24 10 2019].

[24] R. Martin, (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

[25] F. Devos, (2004). Patterns and Anti-Patterns in Object-Oriented Analysis. [Online]. Available: http://www.cs.kuleuven.ac.be/publicaties/doctoraten/cw/CW2004_06.pdf. [Accessed 2020].

[26] SAE International, (1996). Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, SAE ARP4761.

[27] SAE International / EUROCAE, (2010). Guidelines for Development of Civil Aircraft and Systems, ARP-4754A.

[28] F. D. Giraldo, (2015). Introduction to Model-Driven Engineering.

[29] M. Handley, (2006). SDP: Session Description Protocol, RFC 4566, IETF.

[30] ISO, (2018). Information technology - Security techniques - Information security management systems - Overview and vocabulary, BS EN ISO/IEC 27000:2018.

[31] M. Endsley, (1988). Design & Evaluation for Situation Awareness, Proceedings of the Human Factors Society 32rd Annual /Meeting (pp. 97-110), Santa Monica, CA: Human Factors Society.

[32] Ministry of Defence, (February 2017). Safety Management Requirements for Defence Systems, Defence Standard 00-56 Part 2, Issue 5.

[33] A. Baraldi, (2012) ‘Operational Automatic Remote Sensing Image Understanding Systems: Beyond Geographic Object-Based and Object-Oriented Image Analysis (GEOBIA/GEOOIA). Part 2: Novel system Architecture, Information/Knowledge Representation, Algorithm Design and Implementation’ in Remote Sensing Vol 4, issue 9, pp. 2768-2817.

[34] A. Dennis, et al (2015), ‘Systems Analysis and Design: An Object-Oriented Approach with UML’.

[35] L. Liu (2020) ‘Requirements Modeling and Coding:An Object-Oriented Approach’.

[36] CESG, (April 2012). HMG IA Standard No.1 & 2: Information Risk Management, Issue 4.0.

[37] Information technology - Security techniques - Information security risk management, BS EN ISO/IEC 27005:2018.

[38] NIST (April 16, 2018). Framework for Improving Critical Infrastructure Cybersecurity, National Institute of Standards and Technology, Version 1.1. [Online] Available: https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf.

[39] British Standards Institution, (2017). Information technology. Security techniques. Code of practice for information security controls, BS EN ISO/IEC 27002:2017.

[40] British Standards Institution, (2020). Information technology. Security techniques. Evaluation criteria for IT security. Security functional components, BS EN ISO/IEC 15408-2:2020.

[41] María Emilia Cambronero, et al., (2009). 'A Comparative Study between WSCI, WS-CDL, and OWL-S', IEEE International Conference on e-Business Engineering, Volume: 1, Pages: 377-382.

[42] OASIS Standards, (2009). Web Service Atomic Transaction Specification, version 1.2, available at http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec.pdf.

[43] OASIS Standards, (2007). Web Service Business Activity, version 1.1, available at http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-os.pdf.

[44] OASIS Standards, (2007) Web Service Business Process Execution Language, Version 2.0, available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf.

[45] Radio Technical Commission for Aeronautics, (2018). Airworthiness Security Methods and Considerations, RTCA, DO-356A.

[46] International Organization for Standardization, (1989). Information processing systems - Open Systems Interconnection - Basic Reference Model - Part 2: Security Architecture, ISO 7498-2:1989.

[50] The Open Group, (2024). Future Airborne Capability Environment (FACE®) Reference Implementation Guide, Edition 3.0.

FACE® and Future Airborne Capability Environment® are registered trademarks of The Open Group in the United States and other countries.

1 Introduction

1.1 PYRAMID and PYRAMID Reference Architecture

Military aircraft effectiveness is critically dependent on software, especially mission systems software, and fundamental to this effectiveness is the ability to provide new capability where and when it is required. Further to this, effective partnering, capability exchange, and interoperability between allies is essential for operational success.

Traditional software design has been such that relatively small changes can have wide reaching consequences across the aircraft, and the scope for reuse across air platforms (including support systems) and programmes has been limited. This problem has become even more significant with the rapid growth in the complexity of military air system software to meet capability needs. In response, the PYRAMID programme was established to enable technology advantage though systematic software reuse and rapid adaptability.

Modularity and open architectures have been identified as key enablers, but their consistent application across air platforms, and ensuring compatibility with other standards, is essential if the benefits are to be fully realised. In response, a number of open architecture standards have been developed to address areas such as hardware design, data architectures, and software architectures including middleware; but a gap was identified for application software.

The PYRAMID Technical Standard has been developed to provide a consistent approach to modularising air system application software though the PYRAMID Reference Architecture (PRA), whilst ensuring that fundamental requirements, including airworthiness certification and security accreditation, can also be achieved.

This PYRAMID Technical Standard Guidance document has been produced to provide guidance and supporting information to aid understanding and application of the PYRAMID Technical Standard, Ref. [1], enabling the development of PYRAMID compliant systems.

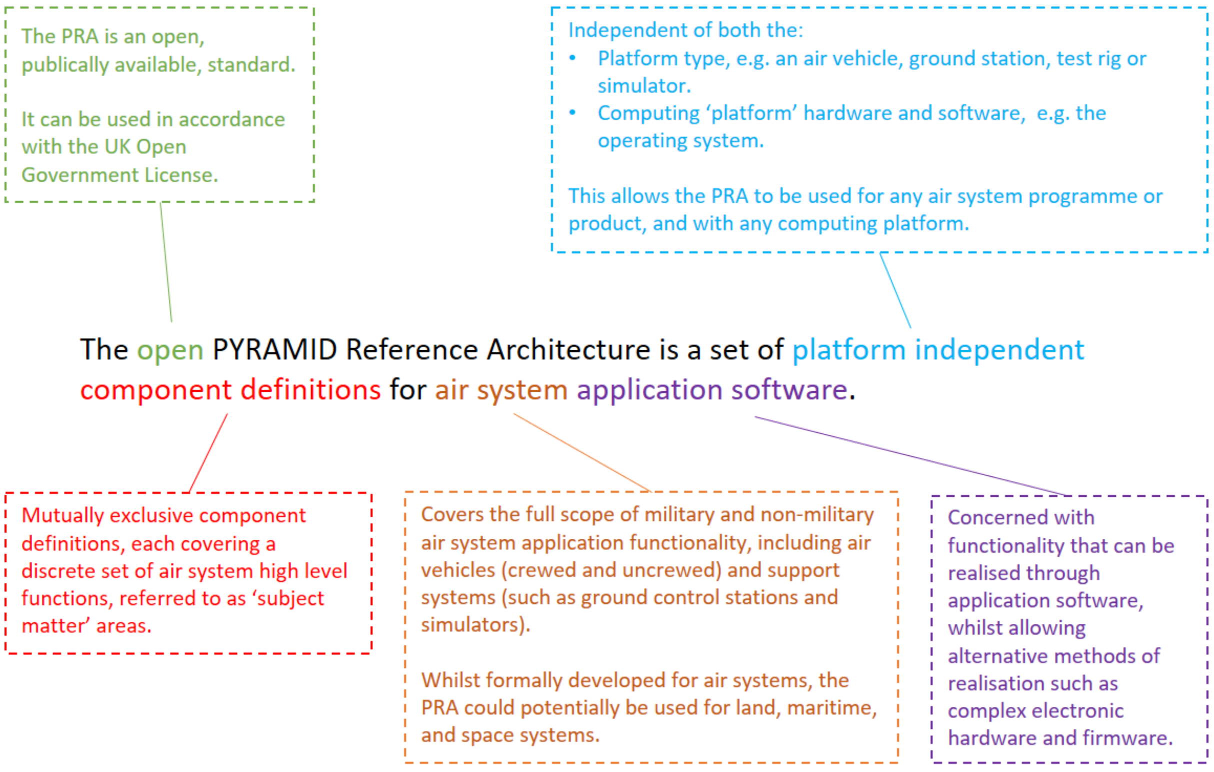

The PRA is an open, publically available, standard.
It can be used in accordance with the UK Open Government License.
Independent of both the:
• Platform type, e.g. an air vehicle, ground station, test rig or simulator.
• Computing 'platform' hardware and software, e.g. the operating system.
This allows the PRA to be used for any air system programme or product, and with any computing platform.
The open PYRAMID Reference Architecture is a set of platform independent component definitions for air system application software.
Mutually exclusive component definitions, each covering a discrete set of air system high level functions, referred to as 'subject matter' areas.
Covers the full scope of military and non-military air system application functionality, including air vehicles (crewed and uncrewed) and support systems (such as ground control stations and simulators).
Whilst formally developed for air systems, the PRA could potentially be used for land, maritime, and space systems.
Concerned with functionality that can be realised through application software, whilst allowing alternative methods of realisation such as complex electronic hardware and firmware.

Figure 1: What is the PYRAMID Reference Architecture?

The PRA defines an open modular architecture for the software aspects of air system functionality. At its core is the decomposition of these software aspects into discrete areas of functionality called PRA components. The PRA component definitions provide the basis from which application software components, with well-defined boundaries, can be developed and integrated into an air system, whether that be an air vehicle or supporting systems. The cohesive and loosely coupled nature of the PRA component decomposition is an enabler to rapid adaptability and reuse across different programmes, air systems, and computing hardware.

The development of the PRA has followed a set of design principles that support the following key goals, with the associated benefits to Exploiting Programmes:

- Exploitability - application across different programmes, air systems, and computing platforms.
- Scalability - the ability to use varying numbers of components, and component variants, to produce air systems and subsystems.
- Utility across a range of mission requirements - the ability to create an air system for various mission scenarios and organisational structures.
- Configurability - the ability to change component behaviour to support the needs of different air systems or missions/operations.
- Flight Certification - a structure that supports the process of certification (and re-certification after system change) in a structured and straightforward manner.
- Security Accreditation - a structure that supports the process of accreditation in a structured and straightforward manner.

- Resilient to Obsolescence - the ability to port system software on to different computing hardware and operating systems with minimal rework.
- Potential for Future Growth - the flexibility for developed systems to adapt to change.
- Supportability - the ability to create reliable and maintainable systems.

1.2 Scope and Purpose

The purpose of this document is to provide guidance on how to understand and apply the PYRAMID Technical Standard, Ref. [1], including associated reference material, such as examples of use.

2 Document Structure

Figure 2: PYRAMID Technical Standard Guidance Document Structure summarises the structure and content of this document, and highlights the content that is generated from a maintained set of UML models, PYRAMID Model, Ref. [3].

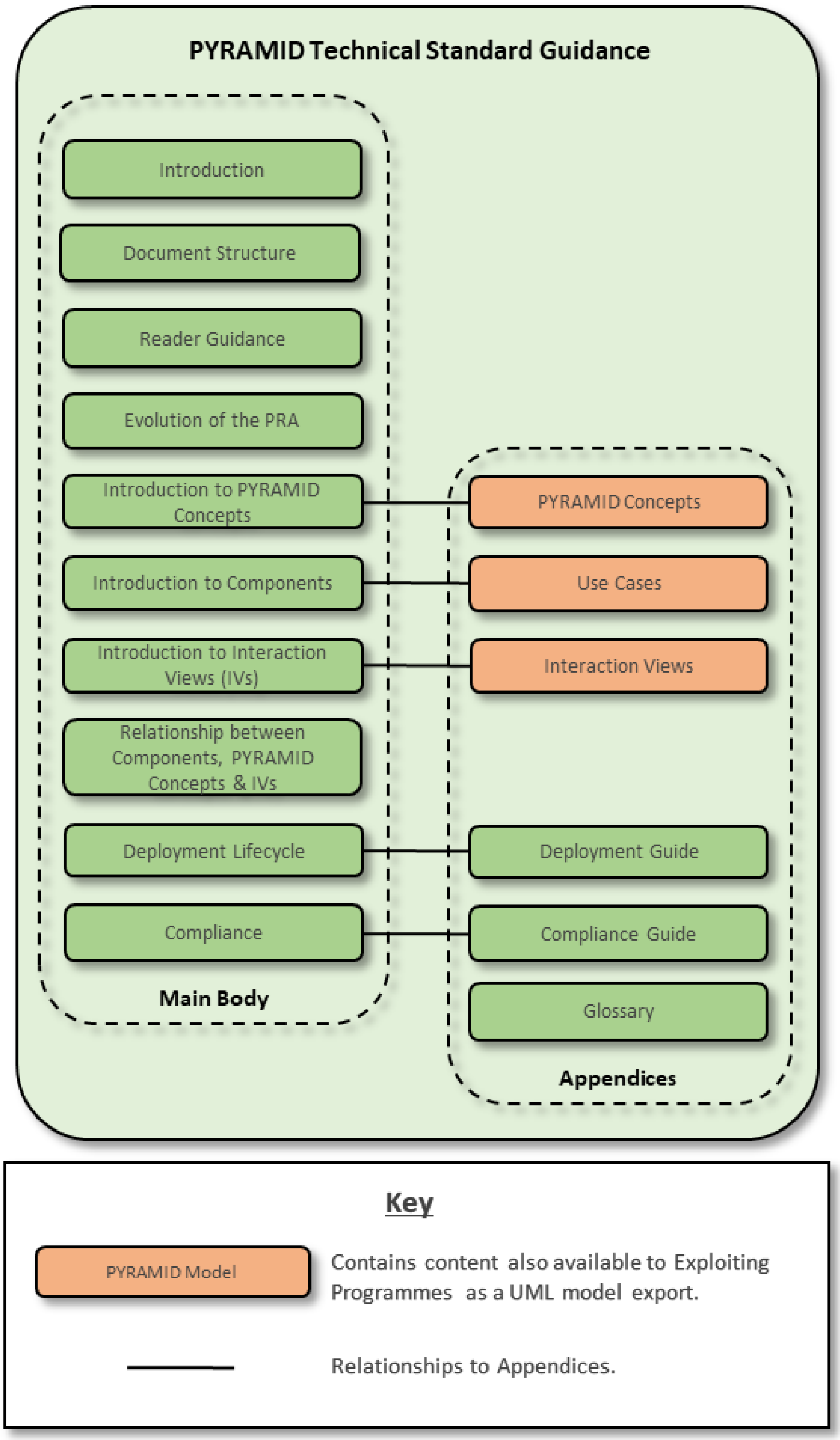

PYRAMID Technical Standard Guidance
Introduction
Document Structure
Reader Guidance
Evolution of the PRA
Introduction to PYRAMID Concepts
Introduction to Components
Introduction to Interaction Views (IVs)
Relationship between Components, PYRAMID Concepts & IVs
Deployment Lifecycle
Compliance
Main Body
PYRAMID Concepts
Use Cases
Interaction Views
Deployment Guide
Compliance Guide
Glossary
Appendices
Key
PYRAMID Model
Contains content also available to Exploiting Programmes as a UML model export.
Relationships to Appendices.

Figure 2: PYRAMID Technical Standard Guidance Document Structure

This document contains the following sections:

- Reader Guidance: Provides guidance on how to read the PYRAMID Technical Standard Guidance document, including its UML content, and introduces some key terms to the reader to aid understanding.
- Evolution of the PRA: Provides historical background regarding the factors that have influenced the development and evolution of the PRA and its associated documentation over time.
- Introduction to PYRAMID Concepts: Discusses a range of concepts, covering a broad cross-section of themes relating to the development of air systems, that have been considered in the development of the PRA and relate to its exploitation. These are described in detail in Appendix A: PYRAMID Concepts, which contains the set of PYRAMID concepts that are intended to assist in understanding the principles that underpin the PRA, and provide guidance on its application enabling consistency across Exploiting Programmes.
- Introduction to Components: Provides an overview of the PRA component set, and the patterns and structure of the PRA component definitions. This information is supported by Appendix B: Use Cases, which defines a set of use cases to assist understanding of how the PRA components may interact to deliver a particular capability, based on the PRA component services defined in the PYRAMID Technical Standard, Ref. [1], section Component Definitions.
- Introduction to Interaction Views: This section introduces interactions views and describes their purpose and usage. Appendix C: Interaction Views, contains a detailed set of example interaction views used to illustrate how the PRA component set could be integrated to achieve a range of system behaviours/functionality that an Exploiting Platform could be expected to provide.
- Relationship between Components, PYRAMID Concepts and Interaction Views: Describes the relationship between PYRAMID concepts, components and interaction views, and how these may be utilised by Exploiting Programmes.
- Deployment Lifecycle: Outlines how PRA components might be developed to produce PYRAMID compliant software components and how these may be deployed as part of a system. Further detail is provided by Appendix D: Deployment Guide, which describes system and software design strategies that could be adopted when using the PRA to develop air systems. It identifies at what stage in the design and development lifecycle artefacts within the PRA and PYRAMID Technical Standard Guidance document can be utilised by an Exploiting Programme to achieve the intended PYRAMID benefits.
- Compliance: This section introduces the guidance on PYRAMID compliance. Further information is provided in Appendix E: Compliance Guide, which discusses the goal of PYRAMID compliance and provides supporting guidance material in relation to the PYRAMID Technical Standard compliance rules, including how compliance with the rules might be achieved, demonstrated and assessed.
- Appendix F: Glossary: Provides definitions of the terms and abbreviations used within the PYRAMID Technical Standard and PYRAMID Technical Standard Guidance document.

3 Reader Guidance

This section introduces some key terms to the reader to aid understanding and provides guidance on how to read the PYRAMID Technical Standard Guidance document.

3.1 Key Terms

Although a full set of glossary terms is provided in Appendix F: Glossary, an appreciation of the following key terms is essential to understanding the PYRAMID Technical Standard Guidance document:

- Deployment: A set of hardware and software elements forming a system (or part thereof) that satisfy the overall system requirements.
- Execution Platform: The infrastructure supporting the execution, communication, etc. of application functionality, e.g. ECOA, ARINC 653, Linux, Windows, and the computing hardware.
- Exploiter: An organisation involved in the design and development of PYRAMID components or the design of PYRAMID deployments.
- Exploiting Platform: A product (e.g. an air vehicle, ground station, or a test rig) that incorporates a PYRAMID deployment.
- Exploiting Programme: A programme developing or incorporating PYRAMID components or a PYRAMID deployment.
- PRA component: A PYRAMID reference artefact, defined by a role, a distinct set of responsibilities, entities and services, for a specific, discrete area of subject matter.
- PYRAMID component: A component that is intended to comply with a PRA component definition.

The term ‘component’ refers to a PRA component or PYRAMID component. Where not explicitly stated, the surrounding text will provide context to identify usage.

The term ‘system’ is used in a variety of ways depending on the context. The system of focus may range in size and scope; from a system of systems (such as multiple air vehicles and supporting assets) to a small sub-system or equipment within a larger system (such as a sensing sub-system or a tactical sensor). It may comprise a variety of hardware (e.g. computing, structural, mechanical and electrical hardware) and software (e.g. PYRAMID application software, other application software, middleware and an operating system).

Since the PRA is designed to be scalable, it is rarely possible to be specific about the size or scope of the system in question. However, within this document it is generally helpful to think of the system as a full military air system, e.g. an air vehicle and, if relevant, an associated ground station. The main exceptions are within the Deployment Lifecycle section, Appendix D: Deployment Guide and Appendix E: Compliance Guide, which refer to the entity being developed by a specific developer, which will be different for different developers working on the same Exploiting Programme.

When referring to a PYRAMID system, or a similar terminology, this refers to a system containing PYRAMID components, typically with the assumption that most or all of the application software comprises PYRAMID components.

In summary, while the term ‘system’ has a typical usage in some areas of the documentation, as described above, the appropriate usage should be determined from the context within which it is written.

3.2 How to read the PYRAMID Technical Standard Guidance document

Figure 3: Recommended PYRAMID Technical Standard Guidance Document Reading Order provides a detailed breakdown of the sections that comprise the PYRAMID Technical Standard Guidance document. It indicates which sections are introductory material, which are reference material, and provides a recommended order in which to read them.

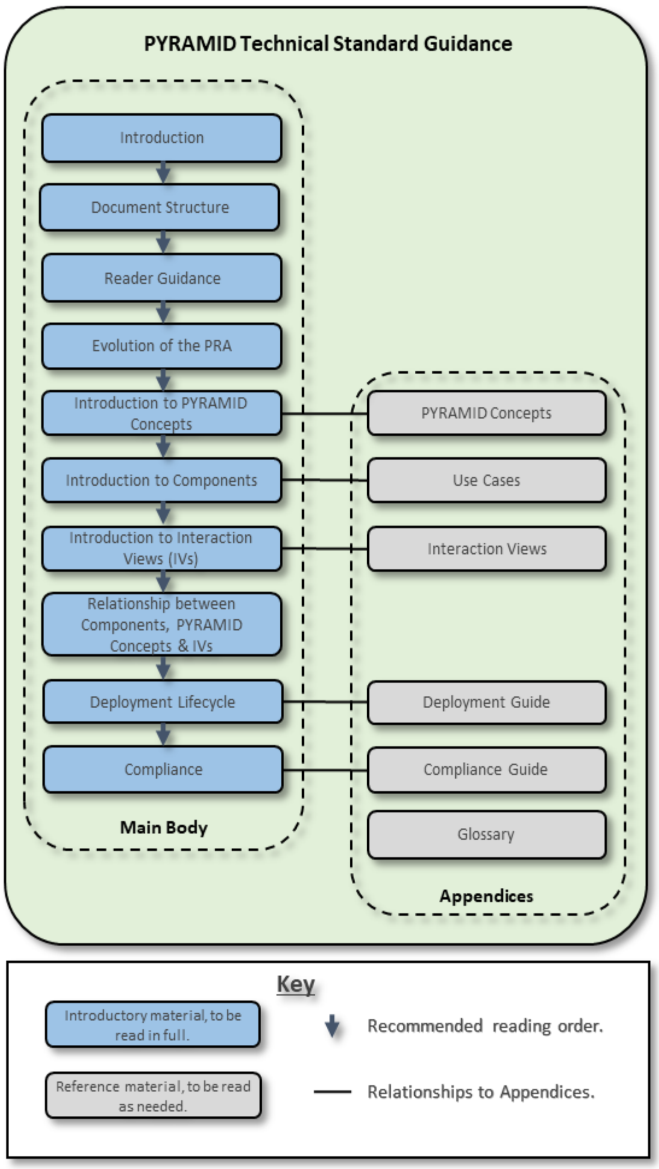

PYRAMID Technical Standard Guidance
Introduction
Document Structure
Reader Guidance
Evolution of the PRA
Introduction to PYRAMID Concepts
Introduction to Components
Introduction to Interaction Views (IVs)
Relationship between Components, PYRAMID Concepts & IVs
Deployment Lifecycle
Compliance
Main Body
PYRAMID Concepts
Use Cases
Interaction Views
Deployment Guide
Compliance Guide
Glossary
Appendices
Key
Introductory material, to be read in full.
Reference material, to be read as needed.
Recommended reading order.
Relationships to Appendices.

Figure 3: Recommended PYRAMID Technical Standard Guidance Document Reading Order

Due to the technical nature of the PYRAMID Technical Standard and PYRAMID Technical Standard Guidance document, they are not necessarily intended to be read in their entirety by any single reader, and as such different readers may wish to read different sections.

Whilst some of the content of the PYRAMID Technical Standard Guidance document describes technical concepts, the main body contains material intended to be accessible to less technical readers, allowing such readers to gain an appreciation of the essence of the PRA and it's supporting information regardless of technical expertise.

The intended audience of the PYRAMID Technical Standard and the supporting PYRAMID Technical Standard Guidance document includes, system designers involved with the design and development of compliant PYRAMID components, system integrators involved in wider integration of PYRAMID compliant systems and any reader who is interested in gaining a broad top level understanding of PYRAMID and the PRA scope.

For more information on Exploiter roles, see Appendix D: Deployment Guide, section How to Read the Deployment Guidance, which suggests the specific sections different Exploiters may wish to focus on within the Deployment Guide.

Note: Much of the content within the PYRAMID Technical Standard and PYRAMID Technical Standard Guidance document is of a detailed technical nature, therefore it is beneficial to have a level of knowledge in:

- The military air system subject matter areas covered by the PRA.
- Systems engineering processes.
- Safety and security-related aspects of engineering.
- Model-Based Systems Engineering (MBSE).
- Reading and understanding Unified Modelling Language (UML) notation.
- An understanding of Service Oriented Architecture (SOA).

References throughout this document to content defined in the PYRAMID Technical Standard, Ref [1], are shown as (non-hyperlinked) green text.

3.3 UML Notation

The PYRAMID Technical Standard Guidance document includes content which is defined and maintained within the PYRAMID Model, Ref. [3], as illustrated in section Document Structure, Figure 2: PYRAMID Technical Standard Guidance Document Structure. This content is also available to Exploiting Programmes in a UML model export format. The PYRAMID model uses UML but it is not a UML software design. It therefore does not adhere rigidly to all UML rules or common conventions. Furthermore, some UML artefacts are used to represent something different to what they would normally be used to represent within a UML software design. It must be emphasised that these deviations are deliberate in order to articulate information in a clear way.

4 Evolution of the PRA

The PYRAMID Programme commenced in 2014 and has resulted in the development of the PYRAMID Reference Architecture (PRA) whose applicability covers military air systems. This was formally documented and published in 2020 as Issue 1 of the PYRAMID Exploiter's Pack. The following years saw the PRA incrementally developed, resulting in annual publications to industry of the PYRAMID Exploiter's Pack, the latest being Version 4.1, Ref. [4]. This was then followed by a standardisation activity, whose aim has been to ensure that PYRAMID and the PRA is more widely adopted, supporting the contracting of future military air systems. It saw further clarification and re-scoping of the PRA definition and content, transforming the PRA and the wider PYRAMID Exploiter's Pack into Issue 1 of the PYRAMID Technical Standard, Ref. [1], and PYRAMID Technical Standard Guidance document, which now supersede all previous issues of the PYRAMID Exploiter's Pack.

Figure 4: PRA Evolution provides a high-level illustration of the evolution sequence of the PRA, which involved:

- Identification of military air system requirements based upon an assumed operational context (comprising all mission phases, mission types and platform types) and conservative safety and security analyses. The requirements were comprehensive and forward looking, taking account of any credible future operational needs.
- Analysis of subject matter domains based upon military air system conceptual design and requirements analysis.
- Identification and maturation of PRA component definitions, PYRAMID concepts (previously known as policies) and interaction views.

During the identification and maturation of the PRA:

- Component subject matter coverage and boundaries were clearly defined and scoped appropriately ensuring their correct level of abstraction.
- Components subject matter and scope were assessed to illustrate indicative safety and security analysis.
- The aims of PYRAMID were considered, and a key set of design principles followed, to ensure realisation of the PYRAMID benefits.

Whilst this body of work has informed the PRA definition (which comprises of the PRA component set), the PRA places no reliance on it and does not directly refer to it.

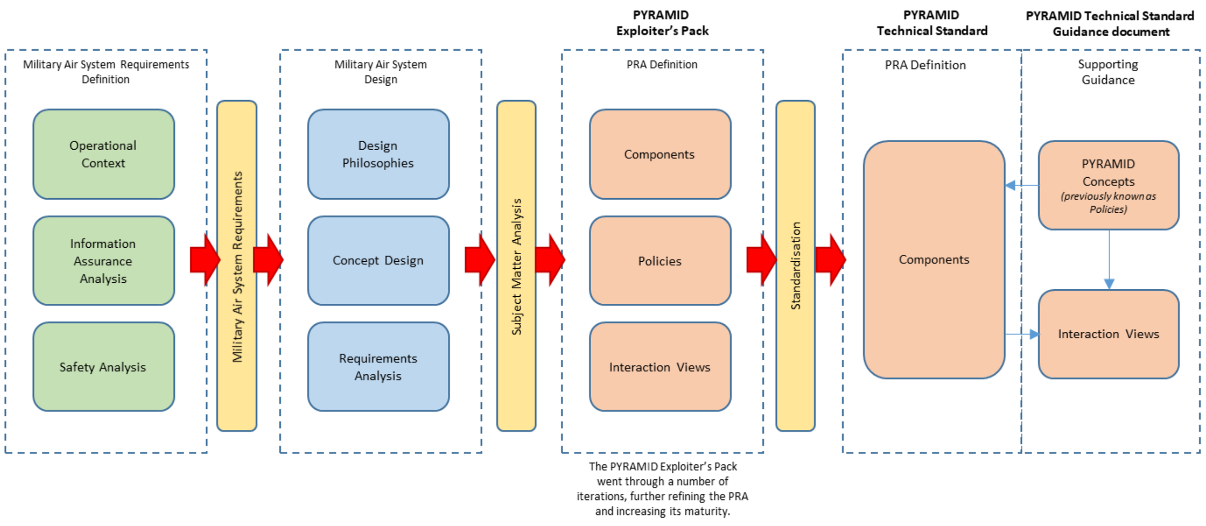

PYRAMID Exploiter's Pack
PYRAMID Technical Standard
PYRAMID Technical Standard Guidance document
Military Air System Requirements Definition
Operational Context
Information Assurance Analysis
Safety Analysis
Military Air System Requirements
Military Air System Design
Design Philosophies
Concept Design
Requirements Analysis
Subject Matter Analysis
PRA Definition
Components
Policies
Interaction Views
Standardisation
PRA Definition
Components
Supporting Guidance
PYRAMID Concepts (previously known as Policies)
Interaction Views
The PYRAMID Exploiter's Pack went through a number of iterations, further refining the PRA and increasing its maturity.

Figure 4: PRA Evolution

The standardisation activity means that the PYRAMID Technical Standard, Ref. [1], now embodies the PRA, and the associated PYRAMID Technical Standard Guidance document contains the majority of the remaining content of the previous PYRAMID Exploiter's Pack, Ref. [4]. Specifically, as can be seen in Figure 4: PRA Evolution, within the PYRAMID Exploiter's Pack the PRA encompassed the PRA component set, policies and interaction views. Within the PYRAMID Technical Standard, the PRA comprises the PRA component set alone while the PYRAMID concepts (previously known as policies within the PYRAMID Exploiter's Pack) and interaction views are provided as supporting guidance material within this document.

5 Introduction to PYRAMID Concepts

This section discusses a range of concepts that have been considered in the development of the PRA and that relate to the exploitation of the PRA.

The PRA itself has been developed from consideration of a broad cross-section of themes and concepts relating to air systems and the application of a number of key architecture design principles. Successful exploitation of the PRA, to develop PYRAMID compliant systems, relies on an understanding of these concepts and principles. The appropriate application of a number of additional design and modelling principles and an understanding of safety and security considerations is also required. These concepts have been categorised as follows:

- **PRA Design Principles:** Principles that have been followed during the development of the PRA and that should be maintained during the development of PYRAMID compliant systems.
- **Air Systems Concepts:** Themes relating to the development of air systems that have been considered during the development of the PRA.
- **Safety and Security Considerations:** How safety and security have been considered in the development of the PRA.
- **PRA Exploitation Principles:** Design principles that are advantageous in achieving the full benefits of the PRA.

The PRA design principles are discussed in section PRA Design Principles. The remaining concepts are introduced in the subsequent sub-sections and discussed in detail within Appendix A: PYRAMID Concepts.

PYRAMID Concept Content Overview

Each PYRAMID Concept section of Appendix A: PYRAMID Concepts includes the following content:

- **Definition:** A description of the concept and why it is important.
- **Design Decisions:** An explanation of how the considerations relating to the particular theme or concept have been applied to the development of the PRA or relate to application of the PRA. This includes key design decisions that have been made during the development of the PRA. This is typically expressed in two key ways:
    - One or more specific PRA components may be defined whose subject matter supports a defined role related to the PYRAMID concept.
    - PRA component responsibilities and services may be defined as being required to support the PYRAMID concept. These may be specific to an individual component or form part of a generalised pattern as described in the PYRAMID Technical Standard, Ref. [1], section Component Composition.
- **Guidance and Examples:** To assist Exploiters in understanding the philosophies that underpin the PRA, and how it is intended to be deployed.
- **Exploitation Considerations:** To identify considerations that are a matter for individual Exploiting Programmes, i.e. for which a particular approach is not defined by the PRA. Since many of the PYRAMID concepts are concerned with architectural principles, technical implementation detail is omitted where this is the responsibility of Exploiting Programmes to define and apply.

Each PYRAMID concept section includes a summary table, the purpose of which is to identify the specific details of how the concept is addressed within the PRA and also those considerations that are a matter for individual Exploiting Programmes. The PYRAMID concepts are detailed in full in Appendix A: PYRAMID Concepts, which also provides more detailed information on the structure and content of each section.

Noting that the PYRAMID concepts are reflected in the PRA component definitions, as described above, Appendix A: PYRAMID Concepts is provided as background and guidance information. For guidance on how to comply with the PYRAMID Technical Standard, Ref. [1], please see Appendix E: Compliance Guide.

5.1 PRA Design Principles

This section describes the key principles behind the design of the PRA and the way in which it is expected to be used. Other principles are described elsewhere within Appendix A: PYRAMID Concepts and Appendix D: Deployment Guide.

The PRA has been designed around the concept of separation of concerns, where different ‘concerns’ are separated and catered for in different ways. In some cases the PRA defines where the separation should be made, and in other cases, in order to maximise its utility and compatibility, the PRA is deliberately independent of some aspects that are separated.

Understanding the separation of concerns is important because correctly taking account of them in an Exploiting Programme helps to ensure that the benefits of PYRAMID are achieved and one of the ‘concerns’ forms the basis on which PYRAMID compliance is established.

Additional design principles that should be applied in an Exploiting Programme can also be found in section PRA Exploitation Principles, which describes design and modelling principles that are advantageous in achieving the full benefits of using the PRA.

5.1.1 Separation of Concerns Defined by the PRA

The foundation of the PRA, and the basis of PYRAMID compliance, is the separation of subject matter:

- **Separation of subject matter:** Domain modelling has been used to separate the PRA into components that each focus on a specific area of knowledge. Each component represents a discrete area of subject matter that includes data associated with a specific area of knowledge and the behaviours that operate on the data associated with this knowledge. A component’s role and responsibilities are defined using terms appropriate to its subject matter, and provide a common language for developers and users of components. The separation of subject matter supports effective and efficient reuse of PYRAMID components between deployments without undue constraint on their design, while the definition of highly cohesive and loosely coupled components minimises the impact of change.

5.1.2 Separations between the PRA and an Exploiting Programme

In order to maximise its utility and compatibility, the PRA separates the following concerns:

- **Separation of 'what' from 'how':** The PRA defines what a component is allowed to do. The detail of how this is achieved, is left to the Exploiting Programme to determine. This includes:
    - The extent to which a PYRAMID component satisfies the PRA component responsibilities and 'how well' they are satisfied, allowing for the appropriate balance between design complexity verses PYRAMID component performance and range of utility (such as the mission scenarios that it can support).
    - The separation of a PRA component across multiple PYRAMID component variants.
    - The instantiation of multiple PYRAMID component instances.
    - The PYRAMID component design, and modelling and software development paradigms or methodologies. This includes the choice of whether to use a service based approach or not.
    - The wider development process and lifecycle.
- **Separation from programme and system requirements:** The PRA does not assume any particular system requirements for an Exploiting Programme. Instead is designed to be able cater for any potential Exploiting Programme requirements, leaving the Exploiting Programme to achieve its requirements through a subset of the components available in the PRA and potentially only incorporating a subset of any given PRA component's responsibilities.
- **Separation from the execution platform:** The PRA is fundamentally about functions that can be achieved through the use of application software, whilst acknowledging that some such functions can also be achieved by alternative means, such as by using dedicated CEH devices. This means that the PRA component definitions are independent of the execution platform (such as the middleware, operating system, or computing hardware), and can therefore be used for any execution platform. This allows an Exploiting Programme to determine how to make the PYRAMID components operable with the execution platform, including decisions about to what extent, and at what stage in the development lifecycle, PYRAMID components are inherently tied to particular aspects of the execution platform. This separation enables Exploiting Programmes to achieve greater reuse and remove the impact of execution platform obsolescence.
- **Separation from a specific system composition:** The PRA is designed to enable a scalable approach to system composition (which components are use and where they are used), meaning that:
    - The size, complexity and capability of a system can be varied through the choice of which components to include.
    - Capability can be split across multiple interacting systems, or sub-systems, without issue; resulting in PYRAMID components residing in the most appropriate system and, where necessary, the same PYRAMID component or a variants of it residing in multiple interacting systems. The interacting systems may be either physically connected or physically separate, such as an air vehicle and a ground station.

  This is achieved through the PRAs independence from 'programme and system requirements' (described above) and its 'loosely coupled' components, which remove or reduce the impact of change on other PYRAMID components when one PYRAMID component is changed. This is due to the fact that the PRA components do not share overlapping subject matter and are connected through the use of bridges, allowing any ‘sematic gap’ between the components to be overcome by the bridge and the connectivity of different components to be handled by the bridge.

The PRA does not separate the following concerns, since they are inherently related through their associated subject matters, but recognises the potential need to separate them, and potential benefits of doing so, in an Exploiting Programme:

- **Separation of mission phase and temporal considerations:** PRA components incorporate all mission phases and temporal considerations (such as intelligence data acquisition and storage, with subsequent data processing and analysis), since these are all within the same subject matter concern. However, due to the separation of ‘what from how’ (described above), Exploiting Programmes are able to separate these different mission lifecycle and temporal concerns into separate PYRAMID components, potentially incorporated into different dedicated systems (such as a mission planning system). Likewise, an Exploiting Programme can incorporate them into the same PYRAMID component, providing the component with a greater capability and greater reuse across systems.
- **Separation of simulation, emulation and test:** Just as they support all mission phases and temporal considerations (described above), for the same reasons, PRA components also support all modes of operation (planning, normal execution, simulation, emulation and test). Again, Exploiting Programmes are able to separate these different modes of operation into separate PYRAMID components, which can be incorporated into different dedicated systems (such as a simulator). Likewise, an Exploiting Programme can incorporate them into the same PYRAMID component, providing the component with a greater capability and greater reuse across systems.

5.2 Air System Concepts

The following concepts related to the development of military air systems are discussed in Appendix A: PYRAMID Concepts:

- Control Architecture: Describes a framework for a control architecture that enables an Exploiting Programme to implement system-wide control.
- Constraint Management: Explains how a PYRAMID deployment can keep within the constraints that limit its behaviour.
- Dependency Management: Describes how the PRA has been designed to enable dependencies to be managed, and conflicts between them to be resolved, so that mission objectives can be achieved.
- Autonomy: Explains what is meant by autonomy within PYRAMID and how it applies to the PRA, including the relationship with authorisation for carrying out activities.
- Health Management: Explains what is meant by health and how the PRA enables a system to manage situations of reduced health.
- Capability Management: Explains how a PYRAMID deployment assesses its ability to perform its designed functions as internal system factors change.

- Multi-Vehicle Coordination: Describes how different vehicles can interact in a coordinated manner through the use of components, including instances of the same component, deployed across different vehicles.
- Interaction with Equipment: Explains how the PRA has been designed to enable a PYRAMID deployment to interact with external equipment, including determination of equipment capability and control of equipment resources.
- Resource Management: Explains what is meant by resources and how resources can be managed when there are multiple demands on them.
- Operational Support: Explains how the PRA can be used for purposes beyond the execution of a mission. It describes a range of such uses, for instance mission preparation, and the components that would support these.
- Storage: Describes the mechanisms provided by the PRA to enable storage of data, and the interactions of the components with storage facilities provided within a PYRAMID deployment.
- Recording and Logging: Defines the means by which components identify information that is needed for current or future use and therefore the information that needs to be retained through recording or logging. It discusses how a component knows how long information should be retained for and if it is no longer required.
- Cyber Defence: Describes how the PRA can be used to provide the system with a level of security monitoring and protection from unauthorised interactions, including how components should work together to protect against different types of cyber attack.
- Human-Machine Interface: Introduces the HMI components and describes how the HMI components are intended to interact.
- Interfacing with Deployable Assets: Deployable assets are hardware which can be deliberately separated from the exploiting platform during a mission. This section describes interfacing with deployable assets, before and after separation, from a PYRAMID compliant Exploiting Platform.
- Tactical Information: Explains how the PRA supports the handling of sensor data and associated tactical information.
- Test: Defines the ability to support testing which is provided by the PRA, including how components support different types of test at different levels of system capability. The scope of this is restricted to self-testing of a PYRAMID deployment.
- Use of Communications: Explains how communications capability may be used by PRA components and is agnostic to components deployed on the same or different platforms, communicating directly or through the use of the communication infrastructure. This includes how components with differing levels of communications ‘awareness' interact with components which provide communications capability.
- Data Exchange: Explains the exchange of data and information between systems, where at least one of which is PYRAMID compliant, and the interactions of the components which provide distribution facilities to support this.

5.3 Safety and Security

Each Exploiting Programme will be expected to take account of the particular safety and security targets that are applicable to that programme and will be responsible for demonstrating that these targets have been met. The PRA only provides indicative safety and security analysis. How safety and security have been considered in the development of the PRA is described in Appendix A: PYRAMID Concepts, which includes the following sections:

- Safety Analysis - Explains how safety analysis has been applied to the PRA and why. Note that the safety analysis does not place any requirement on an Exploiting Programme.
- Security Approach - Explains the approach to security aspects, how it has been applied to the PRA and why. Note that the security analysis does not place any requirement on an Exploiting Programme.

5.4 PRA Exploitation Principles

This section discusses a number of design and modelling principles that should be applied in exploitation of the PRA if the benefits of PYRAMID are to be fully realised. The principles discussed below are supported by the following three concepts in Appendix A: PYRAMID Concepts:

- Component Connections
- Component Extensions
- Data Driving

5.4.1 Component Connections

An Exploiting Programme will combine components to support system functionality through the use of bridges. As PRA components have been developed to be independent of other PRA components, such that the subject matter of each component is distinct from any other component, bridges are used to close the semantic gaps (acting as translators or intermediaries) between deployed PYRAMID components. This enables components, which may have been developed independently, to exchange information seamlessly. Appendix A: PYRAMID Concepts, section Component Connections provides further explanation and guidance on the use of bridges and how components can be connected in a PYRAMID deployment to produce complex systems.

5.4.2 Design Around Counterparting

Each component contains the data relevant to its subject matter. Different components may therefore reason about the same objects or concepts in the real world, but will view them purely from the perspective of their subject matter. For example, a missile loaded onto a wing is understood in different ways by different components; to one component it is a mass that can be detached, to another it is something that can provide a destructive effect. These different views of the same object or concept are referred to as being different counterparts of the object or concept and should be linked by counterpart relationships provided by bridges.

As relevant data is embodied in components, it does not “flow through” the system in the traditional way. In particular, there is no need for data to “pass through” components where no value is added.

Appendix A: PYRAMID Concepts, section Component Connections provides further explanation and guidance on the concept of counterparting.

5.4.3 Design For Change

A driver behind PYRAMID is that future systems must be much more receptive to change. The PRA expects PYRAMID components to be specialised and configured for an Exploiting Platform by the use of data driving. Appendix A: PYRAMID Concepts, section Data Driving, explains how data sets can be utilised in a PYRAMID deployment to modify component behaviour to cater for different conditions; for example, to cater for different role fit equipment or operational requirements.

PYRAMID components can also be extended to add specialised behaviour. Appendix A: PYRAMID Concepts, section Component Extensions, defines what is meant by a component extension, considers their benefits and provides guidance on their usage.

5.4.4 Avoid Component Duplication

Another approach for creating specialised behaviour is to create a unique component variant, i.e. a PYRAMID component that performs a subset of a PRA component's responsibilities or provides a specialised version of it. There are various circumstances where it might be appropriate to do this, such as for safety or security reasons, as described in Appendix D: Deployment Guide. However, given the ability to specialise component behaviour using data driving and extensions, and the benefits of these approaches, these methods should be a primary consideration for accommodating the variations required.

5.4.5 Interaction at PRA Boundaries

The PRA acknowledges that Exploiters may develop systems where parts of the system capability which could have been provided in accordance with the PRA, use non-PYRAMID components instead. For example, a legacy system may be used to provide datalink functionality rather than create a software defined radio built from PYRAMID components. The boundary of the PRA based elements of the system (the PYRAMID deployment scope) is therefore flexible and specific to an Exploiting Programme. This means that the system elements within the PYRAMID deployment scope will interact with the wider system at different levels of abstraction (or detail) based on the nature and capabilities of the wider system. For example, this could range from low level instructions to a sensor on how to make a measurement, to high level instructions to a sensor equipment assembly to track a target.

Furthermore, PYRAMID components are expected to be used in conjunction with the capabilities provided by the computing infrastructure which includes other software such as middleware and operating systems. Therefore, PYRAMID components can, and should, interact with capabilities provided by the infrastructure, such as accessing mathematical functions, managing security and safety partitions, and managing storage media and storage implementation.

These interactions are not shown in the PRA as they will be specific to the deployment of the PYRAMID components. Access to any of these capabilities must never be for a reason that is beyond the scope of the component's subject matter. This means that these capabilities should not be used in a deployment in this way if they duplicate any part of the subject matter of another PRA component.

However, where the PRA is not used for all aspects of the system development, they may still be accessed via the services defined within the PRA as if interacting with another PYRAMID component.

6 Introduction to Components

The PYRAMID Technical Standard, Ref. [1], section Component Set, defines 73 PRA components each bounding a specific subject matter in terms of what each component knows about and what each component does.

The PRA components definitions have taken account of a broad cross-section of concepts that affect air systems, which are discussed in section Introduction to PYRAMID Concepts and Appendix A: PYRAMID Concepts.

The PRA components (see PYRAMID Technical Standard, Ref. [1], section Component Set) are supported by the component composition (see PYRAMID Technical Standard, Ref. [1], section Component Composition) in two ways:

- It augments the PRA component definitions for any responsibilities and services that are common to most or all components.
- It provides a standard pattern, on which the PRA components are based, with more detailed explanation about these standard pattern elements, principally services and responsibilities.

Further explanation of the standard pattern services is provided in Appendix B: Use Cases, which describes generalised use cases, with associated sequence diagrams, showing how the services are expected to be used to support the interaction between different PRA components.

The component composition responsibilities and services provide a level of commonality across all components, ensuring they can work together. Most components will have many, if not all, of the services and responsibilities listed within the component composition, specialised in terms of their specific subject matter. The applicability of these common features and the degree to which they have been specialised within the component definitions is defined in the PYRAMID Technical Standard, Ref. [1], section Component Definitions. In respect of responsibilities and services, the PRA component definitions therefore include:

- The responsibilities and services defined for that component in the Component Set, plus
- The responsibilities and services defined in the Component Composition that are identified as applicable to that component but have not been specialised within the Component Set definition.

Figure 5: PRA Component Set shows the PRA components. Purely for illustrative purposes, the figure groups the components, to emphasise the scope of the PRA and to aid the reader in quickly identifying components potentially relevant to an Exploiting Programme. The groupings are not intended to provide any basis for system partitioning or component interactions, nor do the groupings imply any intent or restriction on how components may be used.

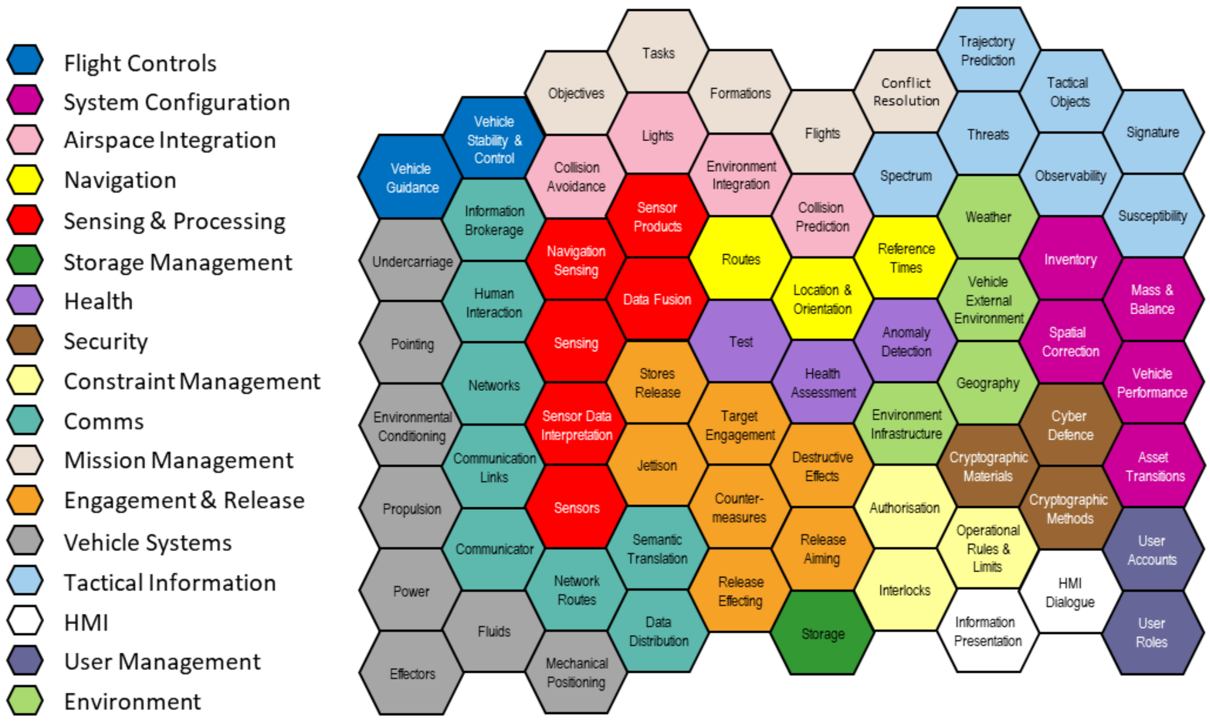


Figure 5: PRA Component Set

7 Introduction to Interaction Views

An interaction view (IV) is used to illustrate how a set of components could be integrated to achieve the system behaviour described by a particular scenario. This IV scenario, together with its accompanying diagram, describes an abstract use of the system and is based on a particular aspect of functionality that an Exploiting Platform could be expected to provide.

System Integrators can use interaction views as examples of combining components to achieve system functionality. Component developers can use interaction views to give examples of context for a component of interest in order to determine potential uses of the component in different deployments.

The developed PYRAMID IVs covering a wide range of mission system functions are provided in Appendix C: Interaction Views.

8 Relationship between Components, PYRAMID Concepts and Interaction Views

As described in section Introduction to Components, the PRA defines a set of components as set out in the PYRAMID Technical Standard, Ref. [1]. This set of components, including many of the features of these components (responsibilities and services), reflect the philosophies described in the PYRAMID concepts as discussed in section Introduction to PYRAMID Concepts. Interaction views then show how a set of components work together to deliver a particular capability, as described in section Introduction to Interaction Views.

Figure 6: The Relationships between PYRAMID Concepts, Components and Interaction Views shows the relationship between PYRAMID concepts, components and interaction views.

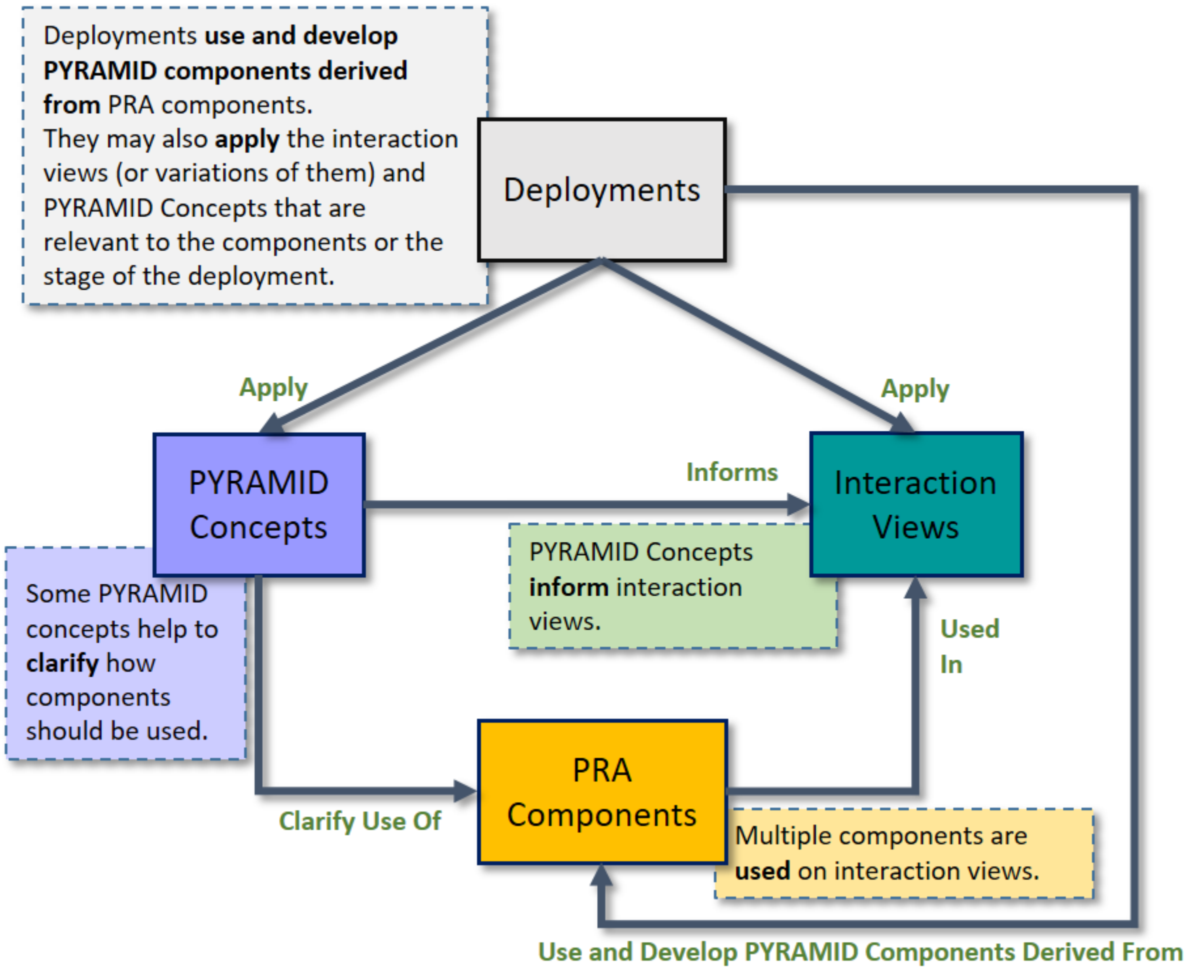

Deployments **use and develop PYRAMID components derived from** PRA components.
They may also **apply** the interaction views (or variations of them) and PYRAMID Concepts that are relevant to the components or the stage of the deployment.
Deployments
Apply
Apply
PYRAMID Concepts
Informs
Interaction Views
PYRAMID Concepts **inform** interaction views.
Some PYRAMID concepts help to **clarify** how components should be used.
Used In
PRA Components
Clarify Use Of
Multiple components are **used** on interaction views.
Use and Develop PYRAMID Components Derived From

Figure 6: The Relationships between PYRAMID Concepts, Components and Interaction Views

Further interaction views, or equivalent architectural design artefacts, may be produced to aid system development across all phases of the mission lifecycle, including planning, execution and post mission analysis. This enables the development of components (within the PRA scope) that support all phases of the mission lifecycle and the deployment of those components, or different component instances or variants, across different systems as necessary. Figure 7: Component and Interaction View Applicability to Mission/Operation Phases provides a view of this.

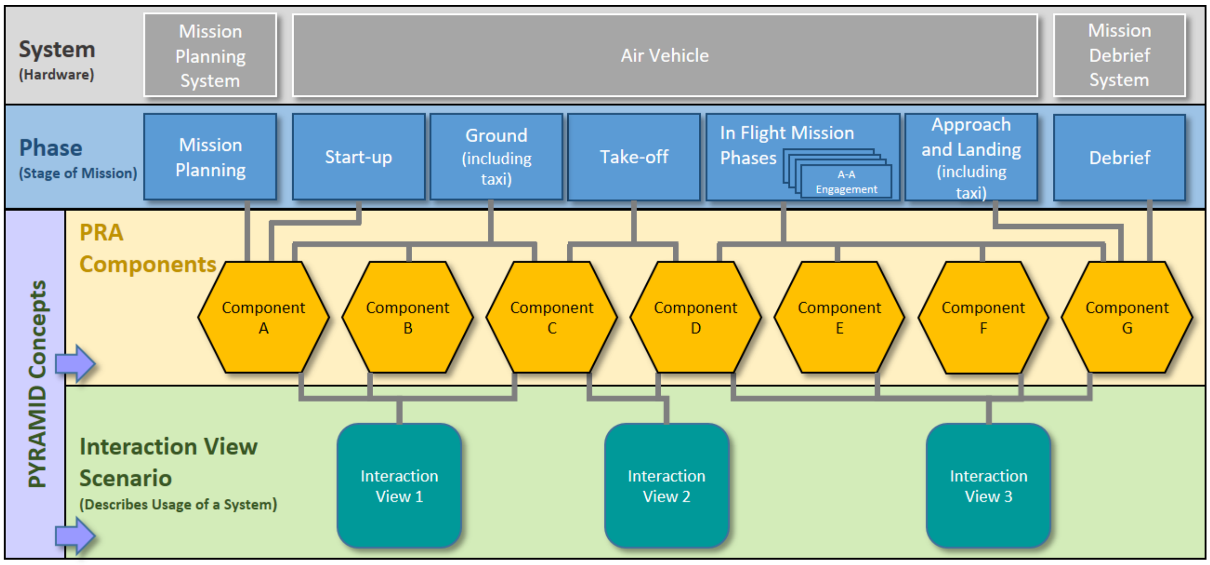
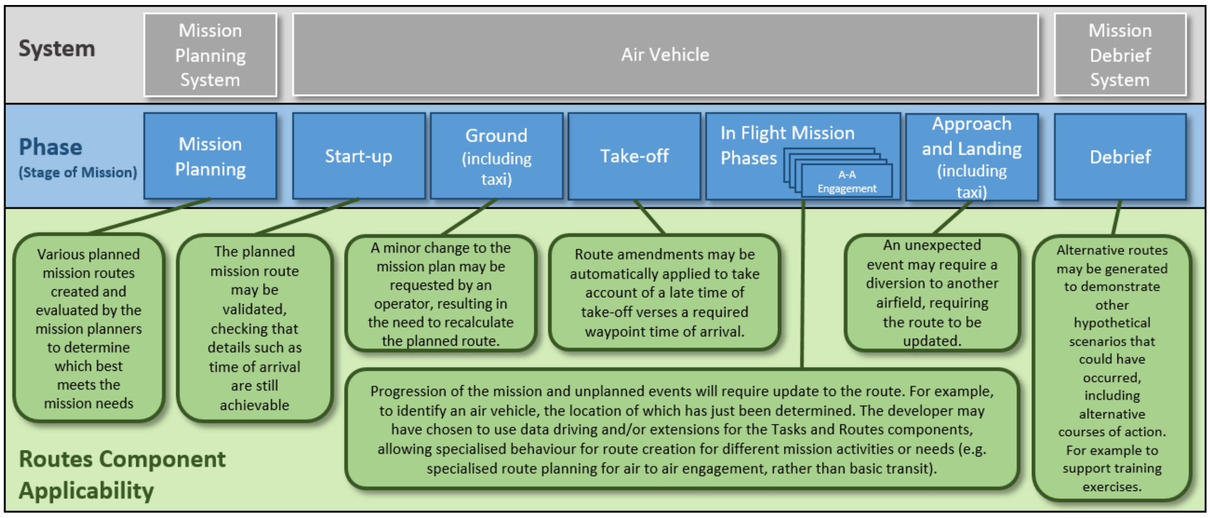


Figure 7: Component and Interaction View Applicability to Mission/Operation Phases

An example of how PRA components are applicable to all mission phases can be seen in Figure 8: The Routes Component's Relevance to Mission/Operational Phases, which shows the applicability of route planning, execution, and replay across mission phases and across different systems. The PRA Routes component would play a dominant role within this by planning and re-planning a route, providing routing demands for the aircraft or an aircraft simulation to follow, and recording the actual route flown for debriefing purposes.



Various planned mission routes created and evaluated by the mission planners to determine which best meets the mission needs

The planned mission route may be validated, checking that details such as time of arrival are still achievable

A minor change to the mission plan may be requested by an operator, resulting in the need to recalculate the planned route.

Route amendments may be automatically applied to take account of a late time of take-off verses a required waypoint time of arrival.

An unexpected event may require a diversion to another airfield, requiring the route to be updated.

Alternative routes may be generated to demonstrate other hypothetical scenarios that could have occurred, including alternative courses of action. For example to support training exercises.

Progression of the mission and unplanned events will require update to the route. For example, to identify an air vehicle, the location of which has just been determined. The developer may have chosen to use data driving and/or extensions for the Tasks and Routes components, allowing specialised behaviour for route creation for different mission activities or needs (e.g. specialised route planning for air to air engagement, rather than basic transit).

Figure 8: The Routes Component's Relevance to Mission/Operational Phases

9 Deployment Lifecycle

This section outlines how PRA components might be developed to produce PYRAMID compliant software components and how these may be deployed as part of a system.

The key terms introduced in this section are:

- Platform Independent Model (PIM): A representation of a system that is independent of the execution platform.
- Platform Specific Model (PSM)**:** A representation of a system that incorporates the execution platform.

Figure 9: Bringing Together PYRAMID Components and Other Software Specifications and Models shows how PYRAMID components might come together with other non-PYRAMID software specifications and be developed into a platform independent design. This includes the reuse of previously developed PIM components, and how these are subsequently developed into something that can be run on a chosen execution platform, again with possible reuse of developed PSM components.

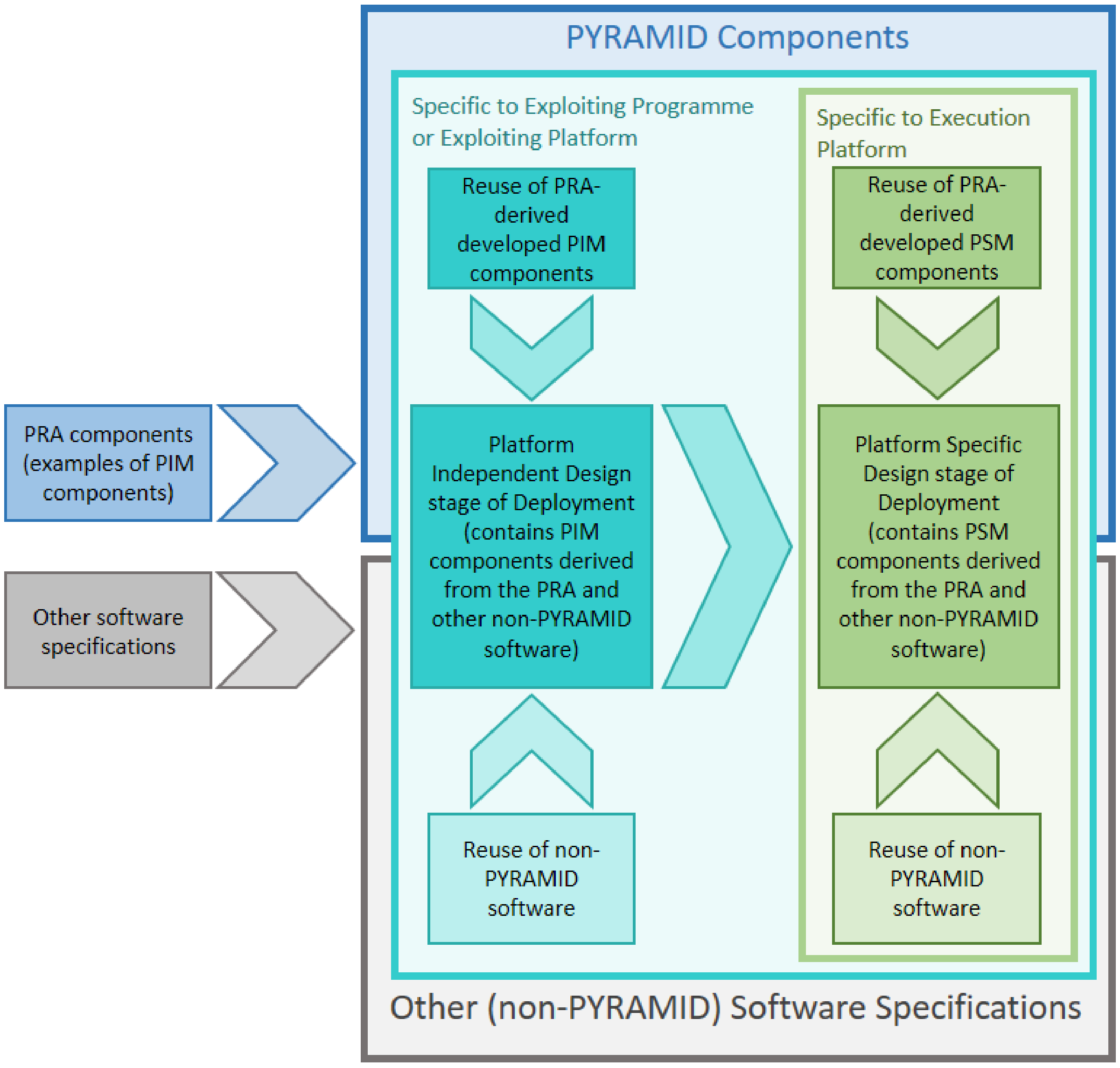

PYRAMID Components
Specific to Exploiting Programme or Exploiting Platform
Reuse of PRA-derived developed PIM components
Platform Independent Design stage of Deployment (contains PIM components derived from the PRA and other non-PYRAMID software)
Reuse of non-PYRAMID software
Specific to Execution Platform
Reuse of PRA-derived developed PSM components
Platform Specific Design stage of Deployment (contains PSM components derived from the PRA and other non-PYRAMID software)
Reuse of non-PYRAMID software
PRA components (examples of PIM components)
Other software specifications
Other (non-PYRAMID) Software Specifications

Figure 9: Bringing Together PYRAMID Components and Other Software Specifications and Models

Figure 10: Deployment Lifecycle Diagram provides a pictorial representation of the PRA and how its contents can be built upon through a deployment process. This section explores the three parts of this figure:

- **The PRA:** As defined by the PYRAMID Technical Standard, Ref. [1].
- **A Platform Independent Design:** A component and/or system design process based on PRA components that does not incorporate execution platform considerations (e.g. is independent of any computation hardware or operating system). A platform independent system design may also incorporate other non-PYRAMID software designs.
- **A Platform Specific Design:** A component and/or system design process developing the Platform Independent Design to incorporate the relevant execution platform considerations into the design. Again, the Platform Specific Design may incorporate other non-PYRAMID software designs.

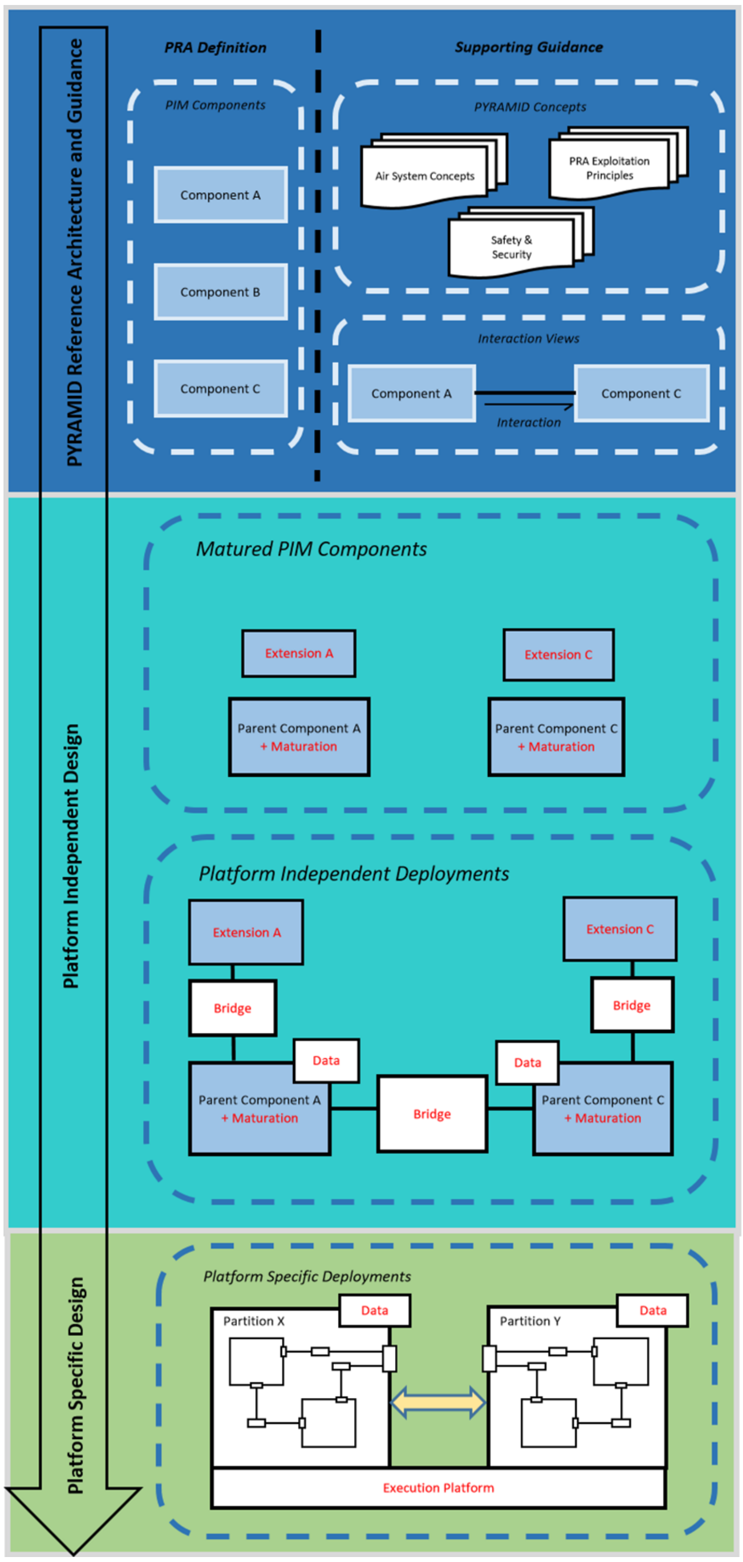


Figure 10: Deployment Lifecycle Diagram

9.1 The PYRAMID Reference Architecture

The PYRAMID Reference Architecture (PRA) is to be used by Exploiters as a starting point to support their development activities and therefore the PRA cannot be modified.

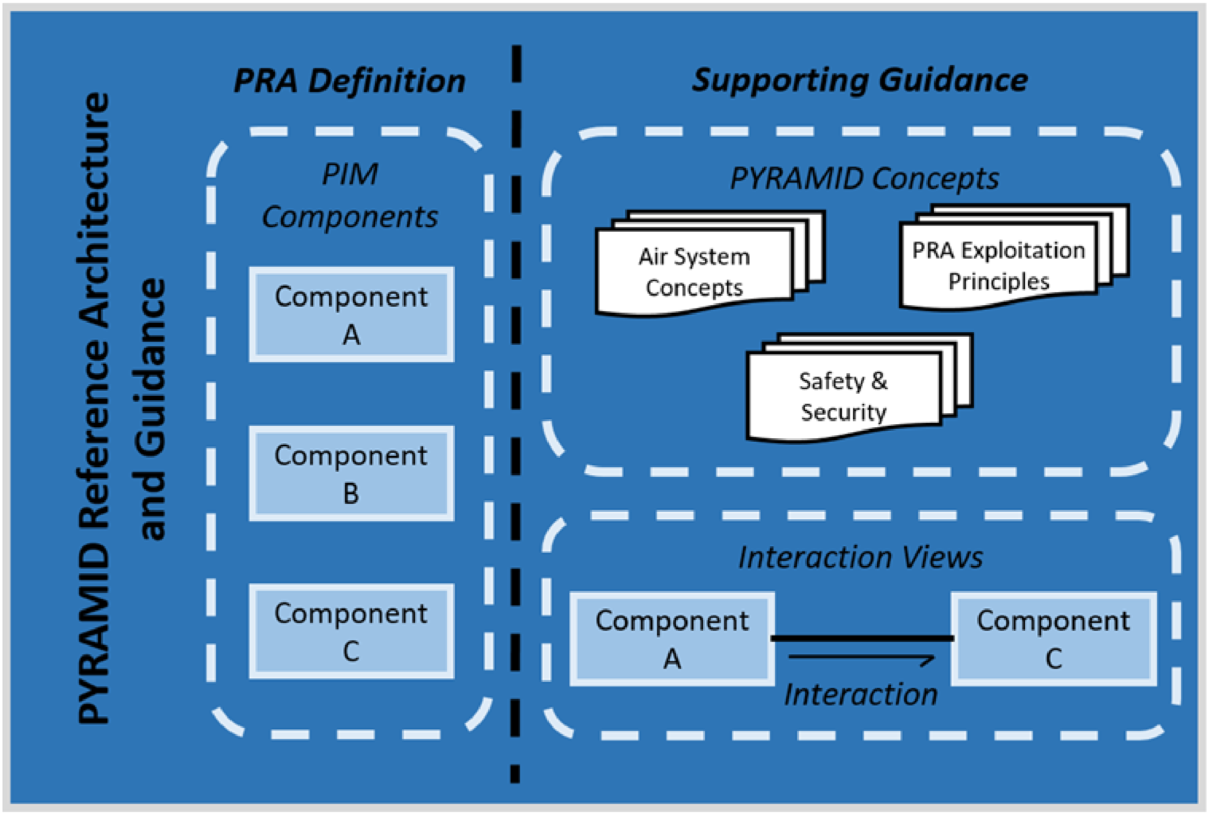


Figure 11: Exploiting Programme use of PYRAMID

A description of PYRAMID concepts, components and interaction views has already been provided. However, Figure 11: Exploiting Programme use of PYRAMID introduces the concept of Platform Independent Model (PIM) components. The components in the PRA are examples of PIM components, meaning that they are independent of any specific Execution Platform. The term “PIM component” is used when describing the development of a component from a PRA component in the platform independent design process, but not all PIM components will be derived from the PRA. Although not shown in Figure 11: Exploiting Programme use of PYRAMID, the Appendix A: PYRAMID Concepts, section Component Extensions, includes some component extensions examples in the form of tactics extensions. Component extensions are discussed further in section Platform Independent Design.

An Exploiter should identify:

- Components relevant to the deployment.
- PYRAMID concepts relevant to the components and to the stage in the deployment lifecycle. Relevant PYRAMID concepts are listed within the component design considerations in the PYRAMID Technical Standard, Ref. [1], section Component Set.

9.2 Platform Independent Design

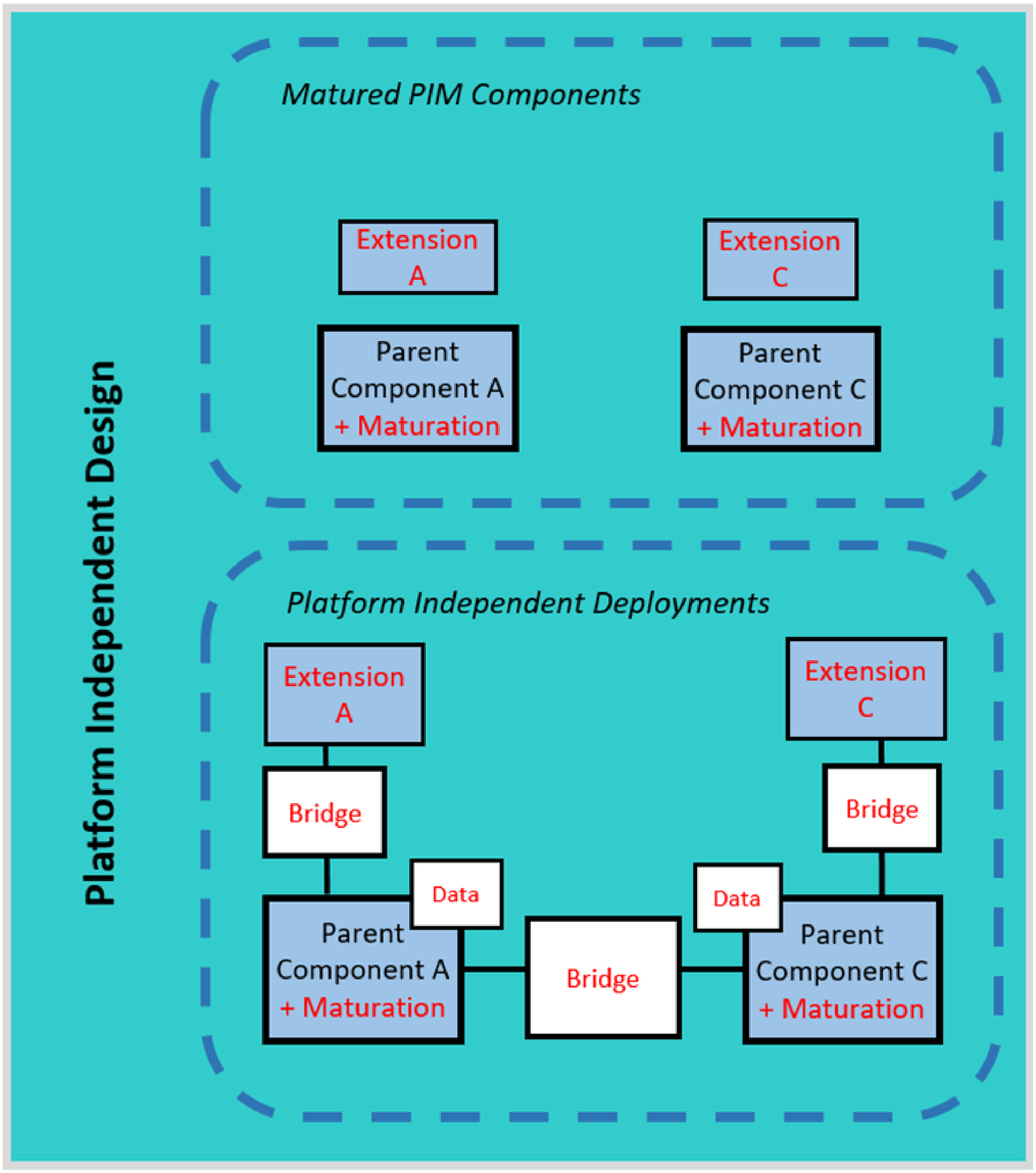

Platform Independent Design
Matured PIM Components
Extension A
Extension C
Parent Component A + Maturation
Parent Component C + Maturation
Platform Independent Deployments
Extension A
Extension C
Bridge
Bridge
Data
Data
Parent Component A + Maturation
Bridge
Parent Component C + Maturation

Figure 12: Platform Independent Design

Figure 12: Platform Independent Design shows how this design process adds further detail and definition to the PIM components derived from the PRA and develops a PIM deployment.

Maturing the PIM component design involves adding the following and in addition Exploiting Programme specific context could be added at this time:

- Class, state and service dependency definitions. (Service dependencies define the relationship between the services required by the component and the services provided by the component).

- Additional component behaviour detail.
- Any subject matter specific data.

A PIM Deployment involves:

- Specifying which PIM components derived from the PRA to use in the PIM deployment.
- Adding bridges and further deployment specific configuration data.

Components within a deployment are independent of each other. This improves the scope for component reuse, minimising the impact of change and easing development maintenance overheads. Simple, self-contained and deterministic bridges are used to connect components together and allow interaction between different parts of a system.

The PRA encourages various methods to provide flexibility in component use and effectiveness, allowing greater adaptability, exploitability, reuse and configurability:

- **Component Extensions:** These allow the components to separate out, extend or specialise the functionality provided by a parent component, with the parent responsible for providing services to other components. This allows component extensions to be developed with a degree of independence.
- **Data Driving:** A method for allowing a component's specific behaviour to be specified or optimised through the use of data files. This reduces the need for component redesign.
- **Multiple Component Variants:** Multiple variants of a component can be used within a deployment in order to allow components to be tailored for specific purposes (e.g. to support different sensor types) or to distribute/compartmentalise behaviour (e.g. for safety or security reasons).

Appendix D: Deployment Guide sections Platform Independent System Design and Platform Independent Component Design provide more detail on how PRA, PYRAMID concepts and IV artefacts can be used in platform independent design.

9.3 Platform Specific Design

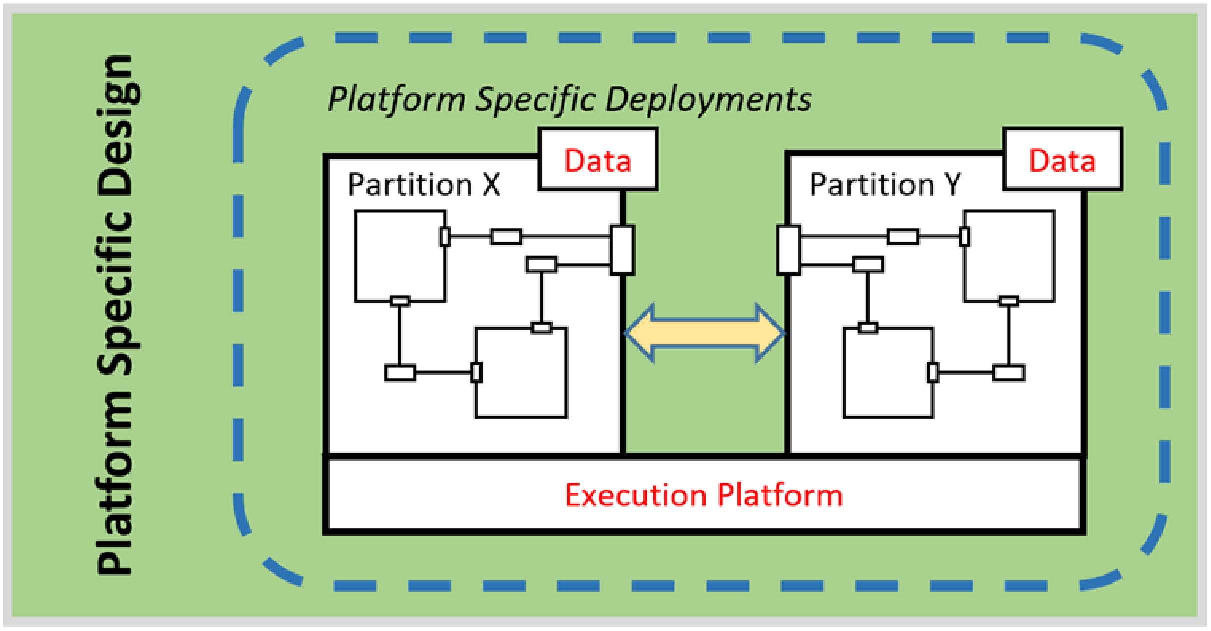


Figure 13: Platform Specific Design

Figure 13: Platform Specific Design shows the result of how this design process might transform the PIM deployment into a Platform Specific deployment by taking account of the execution platform (i.e. being compatible with the software and computing infrastructure that is outside of the PRA scope).

Development activities potentially include satisfying non-functional requirements (such as latency, safety and security), adding implementation specific data, and software implementation requirements (such as data driving, partitioning and coding languages). Partitioning is the separation of software in some way, often via some physical means.

Appendix D: Deployment Guide sections Platform Specific System Development and Platform Specific Component Development provide more detail on how PRA, PYRAMID concepts and IV artefacts can be used in platform specific design.

10 Compliance

This section introduces the guidance on PYRAMID compliance provided in Appendix E: Compliance Guide.

The rules for achieving compliance with the PYRAMID Technical Standard are defined in the PYRAMID Technical Standard, Ref. [1], section How to Comply with the PYRAMID Technical Standard. In order to establish that the PYRAMID approach has been adhered to, the standard defines compliance rules for individual components, component connections and deployments.

Appendix E: Compliance Guide discusses the goal of compliance and provides supporting guidance material in relation to the compliance rules, including how compliance with the rules might be achieved, demonstrated and assessed. While Appendix E: Compliance Guide provides guidance and direction on how compliance might be declared, a formal process for verifying compliance is out of scope of this document. It is expected that any formal process for verifying compliance, including the definition of any associated acceptance criteria, will be separately established for each Exploiting Programme.

It is recommended that developers assess the attained compliance of each component or deployment against the PRA definition throughout the development lifecycle and it is expected that the attained compliance will be required to be understood and declared by the Exploiting Programme at its key programme milestones. It is probable therefore that compliance reviews would be held alongside an Exploiting Platform’s design reviews.

For further information, see Appendix E: Compliance Guide.

Appendix A: PYRAMID Concepts

A.1 Introduction

This appendix defines the PYRAMID concepts that have been considered in the development of the PRA and that relate to the exploitation of the PRA, as described in section Introduction to PYRAMID Concepts.

They are intended to assist Exploiters in understanding the concepts that underpin the PRA, and provide guidance on their application so that they can be applied in a consistent way across a number of Exploiting Programmes.

A.1.1 Structure of PYRAMID Concepts

The basic structure of each PYRAMID concept is as follows:

1. **Pre-Reading and Related PYRAMID concepts** - Identifies key concepts necessary to understand the PYRAMID concept topic**.**
2. **Introduction** - Introduces the PYRAMID concept topic, including:
    a. What the PYRAMID concept is.
    b. Why the PYRAMID concept is important.
3. **Overview** - Summarises the key points of the PYRAMID concept.
4. **Detailed Breakdown** - Subsequent sections that explain the PYRAMID concept in detail.
5. **PYRAMID Concept Summary** - Identifies the specific details of how the PYRAMID concept is reflected in the PRA, and identifies considerations that are a matter for individual Exploiting Programmes for which the PYRAMID concept does not define a particular approach.

Diagrams within the PYRAMID concepts generally abbreviate how service connectivity is achieved. In particular, although connections between component services are always implemented by bridges, these are not shown except where they are the specific focus of the PYRAMID concept.

A.1.2 PYRAMID Concept Summary Table Guidance

Each PYRAMID concept explains how the considerations relating to a particular topic or concept, related to air systems, are catered for within the PRA. Each PYRAMID concept concludes with a summary table, the purpose of which is to identify the specific details of how the PYRAMID concept is reflected in the PRA. This may include the identification of specific component responsibilities or services, or the identification of PRA components that have a specialised role in relation to the PYRAMID concept topic. By providing this mapping from the PYRAMID concepts to the PRA, the summary tables help understanding of how compliance with the PRA defined subject matters, de facto, achieves adherence to the PYRAMID concepts. The PYRAMID concepts themselves, including the summary tables are therefore provided purely as guidance material.

The content of the PYRAMID concept tables is summarised in the diagram below. The information captured falls in to three categories:

1. **General PRA Component Support:** Component responsibilities that are required to support the PYRAMID concept and that are identified as being applicable to multiple PRA components. Column 1 captures the general component behaviours identified in the PYRAMID concept. Column 2 captures the corresponding specific component responsibilities

and services defined within the PYRAMID Technical Standard, Ref. [1], section Component Definitions.

2. **Specific PRA Component Support:** Specific areas of functionality that the PYRAMID concept identifies, related to the concept under consideration, for which the PRA defines a distinct subject matter. Column 1 list the specific areas of functionality identified in the PYRAMID concept. Column 2 list the corresponding components defined in the PRA.
3. **Exploiter Considerations:** Considerations that are a matter for individual Exploiting Programmes for which neither the PRA nor the PYRAMID concept define a particular approach.

Thus, while the first column summarises key aspects of the PYRAMID concept, the second column provides details of how the PYRAMID concept manifests within the PRA, i.e. the specific details of what an Exploiter will find in the PRA that addresses the PYRAMID concept statements. Where there is overlap or a relationship between PYRAMID concepts, reference to other PYRAMID concepts may be provided. This aims to keep the tables concise and avoid repetition.

The tables do not seek to do the following:

1. Repeat definitions of terms that can be found in the main PYRAMID concept text.
2. Repeat any rationale for why a particular approach has been taken or the benefits that this provides.

	PYRAMID Concept Summary	**PRA Standard Approach**
General PRA component support for the PYRAMID concept	What general component behaviours does the PYRAMID concept identify in relation to this topic?	What component responsibilities and services does the PRA define to support the approach defined within the PYRAMID concept? (References may be made to related PYRAMID concepts where applicable.)
Specific PRA component support for the PYRAMID concept	What specific areas of functionality does the PYRAMID concept identify in relation to this topic?	What components does the PRA define with a specialised role related to this topic? (Where PRA components are listed in this cell, they have a one to one mapping to the functional areas identified in the left adjacent cell.)
Exploiter considerations	What considerations related to this topic are a matter for individual Exploiting Programmes?	*The PRA does not mandate a specific approach in respect of these considerations.*

Figure 14: PYRAMID Concept Summary Table Guidance Diagram

It should be noted that while the majority of the PYRAMID concept summary tables follow the format shown in the above diagram, there are several exceptions. The following three PYRAMID concepts relate to general modelling principles:

- Component Connections
- Component Extensions
- Data Driving

In the case of these three PYRAMID concepts, the summary tables capture the key PYRAMID concept statements and identify any exploitation specific considerations. There are however, no related component responsibilities or services, nor any specialised components, associated with these PYRAMID concepts.

A.2 Air System PYRAMID Concepts

Air System concepts are concepts that provide consistent ways of developing a PYRAMID deployment, that support PYRAMID benefits.

A.2.1 Control Architecture

A.2.1.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Dependency Management
- Autonomy
- Human-Machine Interface
- Constraint Management
- Capability Management
- Resource Management
- Interaction with Equipment
- Multi-Vehicle Coordination

A.2.1.2 Introduction and Scope

This PYRAMID concept describes how the PRA has been designed to enable the architect of an Exploiting Platform to implement system-wide control through use of a layered architecture.

A.2.1.2.1 What is the Control Architecture?

A PYRAMID deployment will be given mission objectives that it needs to achieve. The control architecture shows how components could be arranged to work together to achieve these objectives.

The Control Architecture PYRAMID concept defines layers of components which have different roles in the achievement of mission objectives. It shows how a PYRAMID deployment can achieve such objectives by the successive delegation of requirements through the components to coordinate the use of resources.

A.2.1.2.2 Why is the Control Architecture Important?

The PRA aims to be configurable and to support a range of operational scenarios. The control architecture PYRAMID concept plays an important role in achieving the PYRAMID benefits by allowing resources to be used, and coordinated, in the best way to achieve a mission objective, both by allowing a wide range of configurations and by allowing flexible communication between components within a configuration.

The control architecture also increases readability of the PRA for potential PRA exploiters by giving context to the PRA components.

A.2.1.3 Overview

The control architecture PYRAMID concept can be summarised as follows:

- Each component is allocated to a layer which helps bound its role in meeting mission objectives.
- Requirements are delegated through each layer such that decisions can be made at the most appropriate layer, and problems resolved at the lowest possible layer (e.g. the resolution of requirements) while providing overall control.
- The responsibility for and control of resources occurs at the lowest appropriate level of the architecture.
- Within the Resource Layer there are components that deal directly at the lowest level of specific external equipment. Other Resource Layer components can utilise the services these components provide to control specific external equipment and gather information about the real world.

A.2.1.4 Objective Breakdown

By definition, a mission objective is achieved by carrying out a number of tasks. A task is an activity to be carried out in support of a mission objective and is executed via specific actions that coordinate the use of resources of the system. This breakdown is shown in a simple form in Figure 15: Objective Breakdown. For more complex mission objectives more levels of breakdown may be necessary to obtain the solution. For example, it is possible for tasks to be derived from other tasks, actions from other actions, and steps from other steps.

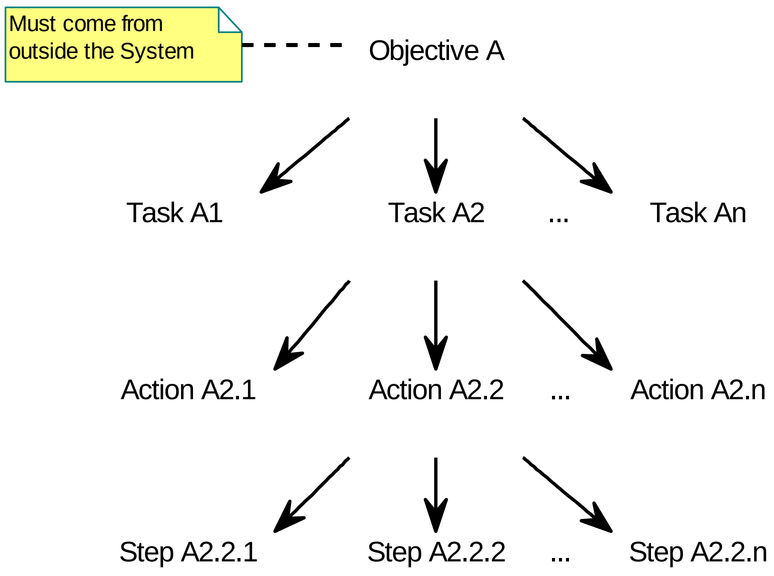
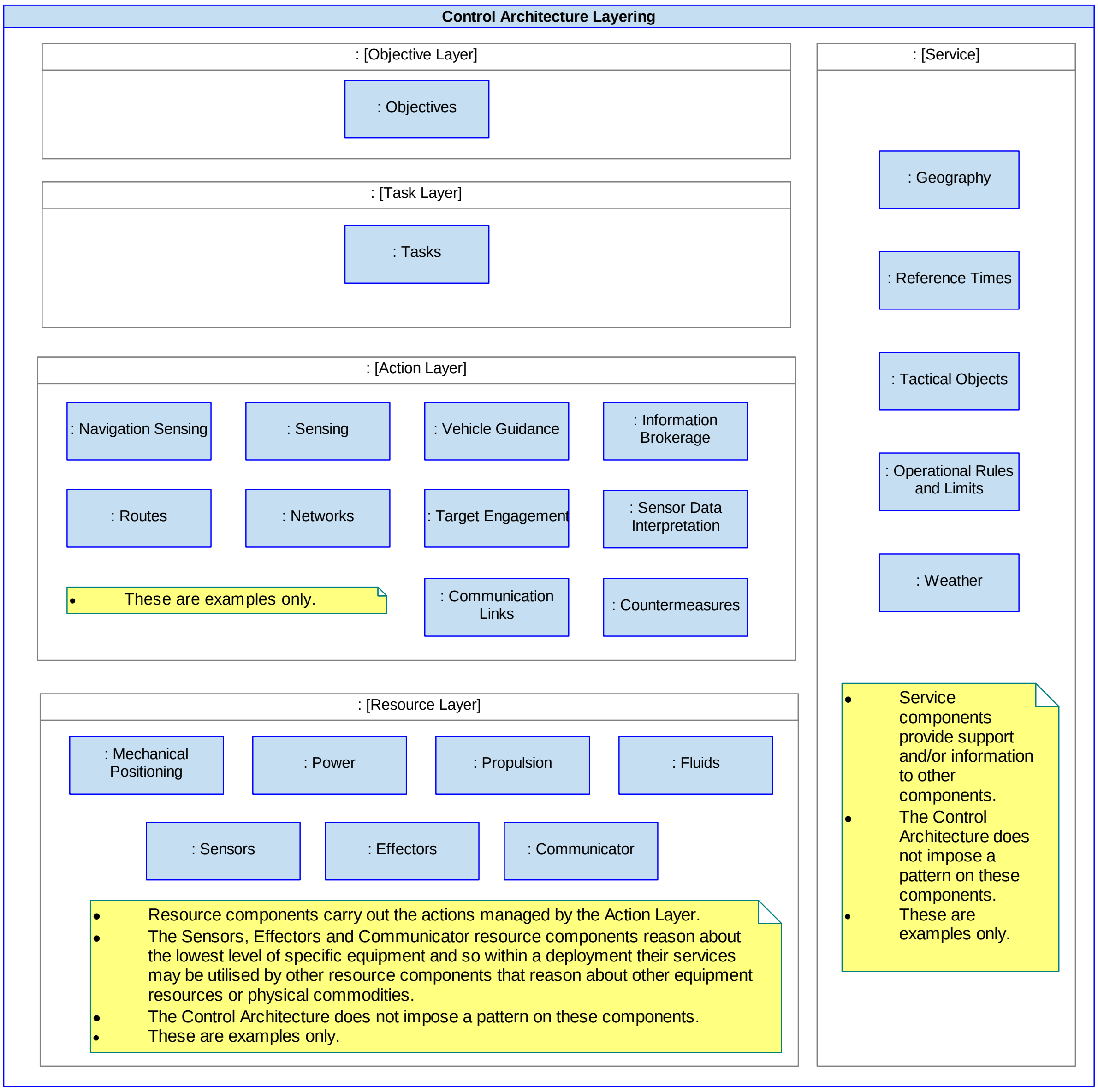


Figure 15: Objective Breakdown

A.2.1.5 Layers

The layers of the control architecture are shown in Figure 16: Control Architecture Layering. The control architecture is arranged into four layers. These layers are based on the principle that mission objectives are met by tasks, which are achieved by carrying out actions, using resources. A deployment with low levels of autonomy is likely to focus on lower layers of the architecture, with simple, or absent, components at the higher layers. Not all components fit into a layer: many components provide information or general services in support of the control architecture. These service components provide services to all layers of the architecture.

Interaction with an external system, standalone equipment, or an operator can be performed with any PRA component. The Interaction with Equipment PYRAMID concept explains how capability and resources provided by equipment external to the system can be integrated at different levels of abstraction in a PYRAMID deployment and accessed by the components at different layers within the architecture. Therefore, in some cases an external system, standalone equipment, or operator may be selected to perform the role (in full or in part) of a PRA component. For example, in less autonomous deployments the operator may fulfil the role of the Objectives component and Tasks component.



- Resource components carry out the actions managed by the Action Layer.
- The Sensors, Effectors and Communicator resource components reason about the lowest level of specific equipment and so within a deployment their services may be utilised by other resource components that reason about other equipment resources or physical commodities.
- The Control Architecture does not impose a pattern on these components.
- These are examples only.

- Service components provide support and/or information to other components.
- The Control Architecture does not impose a pattern on these components.
- These are examples only.

Figure 16: Control Architecture Layering

A.2.1.5.1 Objective Layer

The Objective Layer contains a single component called Objectives. This component will be provided with requirements from outside the system which define the mission objectives and form the mission context. The role of the Objectives component is to manage these mission objectives through the execution of tasks.

A.2.1.5.2 Task Layer

The Task Layer contains a single component called Tasks. The role of the Tasks component is to execute tasks through the coordinated execution of actions. The Tasks component may be supplemented through the use of component extensions. For examples of the Tasks extensions, see the Component Extensions PYRAMID concept. The Tasks component has overall responsibility for achieving tasks placed on it by co-ordinating the execution of actions.

A.2.1.5.3 Action Layer

The Action Layer contains multiple components (see Figure 16: Control Architecture Layering for examples). The role of these components is to perform actions by managing the resources available to them. Each action component addresses actions within a particular area of subject matter. An action component is responsible for identifying a solution which can achieve the requirements given to it. These requirements vary widely, Appendix C: Interaction Views gives many examples. Action components are responsible for identifying any dependencies (e.g. activities or information) that they have in order for their actions to be satisfied. These dependencies could be satisfied by the Tasks component or other action, resource or service components.

A.2.1.5.4 Resource Layer

The Resource Layer contains components that are closely associated with the management of resources in support of the actions that are managed at the Action Layer. These components either reason exclusively about resource equipment or about non-equipment resources (e.g. fuel). The Sensors, Effectors and Communicator resource components reason about the lowest level of specific equipment. Within a deployment, other resource components (that reason about other equipment resources or physical commodities) can utilise the services they provide to directly control equipment, pass data and gather information about the real world. The Sensors, Effectors, and Communicator components will be used within deployments that are developed to fully use PYRAMID components wherever possible.

In accordance with the Interaction with Equipment PYRAMID concept, any component can interact with external equipment or systems where appropriate, such as where the external equipment contains software that performs the function of a PRA component. Therefore, resource components do not provide the exclusive interaction with resources. This PYRAMID concept does not impose any specific conditions or pattern on components in the Resource Layer.

A.2.1.5.5 Service Components

Service components are those components which do not fall into any specific layer of the control architecture. These components can provide information and perform activities to be used by other components requiring that information or support. To do so, service components may consume information and, where required, place dependencies on other components. This PYRAMID concept does not impose any specific conditions or pattern on service components.

A.2.1.6 General Principles

The control architecture results in or promotes the following general principles and guidance:

- Control is distributed throughout the system. There are no central management components, since each component manages the activities that are within its subject matter.
- What a component can and cannot do is determined by its subject matter, not by the control architecture. However, the control architecture helps to define the level of abstraction of components and, therefore, the level of abstraction of requirements that they can satisfy. Whilst the subject matter of each component, including its level of abstraction, will place practical limits on which components can satisfy the requirements of other components, the control architecture does not inherently impose a rigid control hierarchy.
- The control architecture does not restrict interaction between components and unnecessary data transfer through intermediate components should be avoided, including for:
  - Information sharing.
  - Coordination of activities contributing to requirements.
  - Application of constraints.
- Decisions should be made within the appropriate layer by the component with the associated subject matter and therefore knowledge; however, problems (such as conflicting requirements) should be resolved at the lowest layer where it is possible to do so.
- The use of system resources is, directly or indirectly, available to any component that can make use of them, allowing a single system resource to be used for a range of different for purposes.
- User or external system requirements can be placed on a component in any layer of the control architecture as determined from the level of abstraction and subject of the requirement.
- The control architecture can span multiple systems, including multiple vehicles, to enable integrated or compatible systems that are capable of achieving interoperable, coordinated, and holistic solutions. This can require coordination between different instances of the same PYRAMID component or between different PYRAMID components within different systems.
- Some service components contribute to system wide control, potentially working in conjunction with components in any layer of the control architecture. For example:
  - The Operational Rules and Limits component can impose limits on what can be done.
  - The Interlocks component can help to ensure that inappropriate or unsafe actions do not take place.
- It is not necessary for a deployment to implement all layers of the control architecture, or all components within the PRA.

A.2.1.6.1 Breakdown of Objectives by Components

Figure 17: Component Interactions to Breakdown Objectives shows how component interactions correspond to the objective breakdown shown in Figure 15: Objective Breakdown. The Objectives component places task requirements onto Tasks; Tasks places action requirements onto components in the Action Layer; and the action components place requirements on other action components to coordinate and command associated activities. The action components can also issue commands to

resource components to carry out the steps required to fulfil the objective. The resource components can interact with other resource components that deal with specific equipment. Whether something a component requires to happen is treated as a command or coordination is dependent on the deployment design. Note that the diagram shows the interactions through the use of the Solution_Dependency and Requirement generic services, defined within the component composition (see PYRAMID Technical Standard, Ref. [1], section Component Composition). The use of these services is described in more detail in the Dependency Management PYRAMID concept.

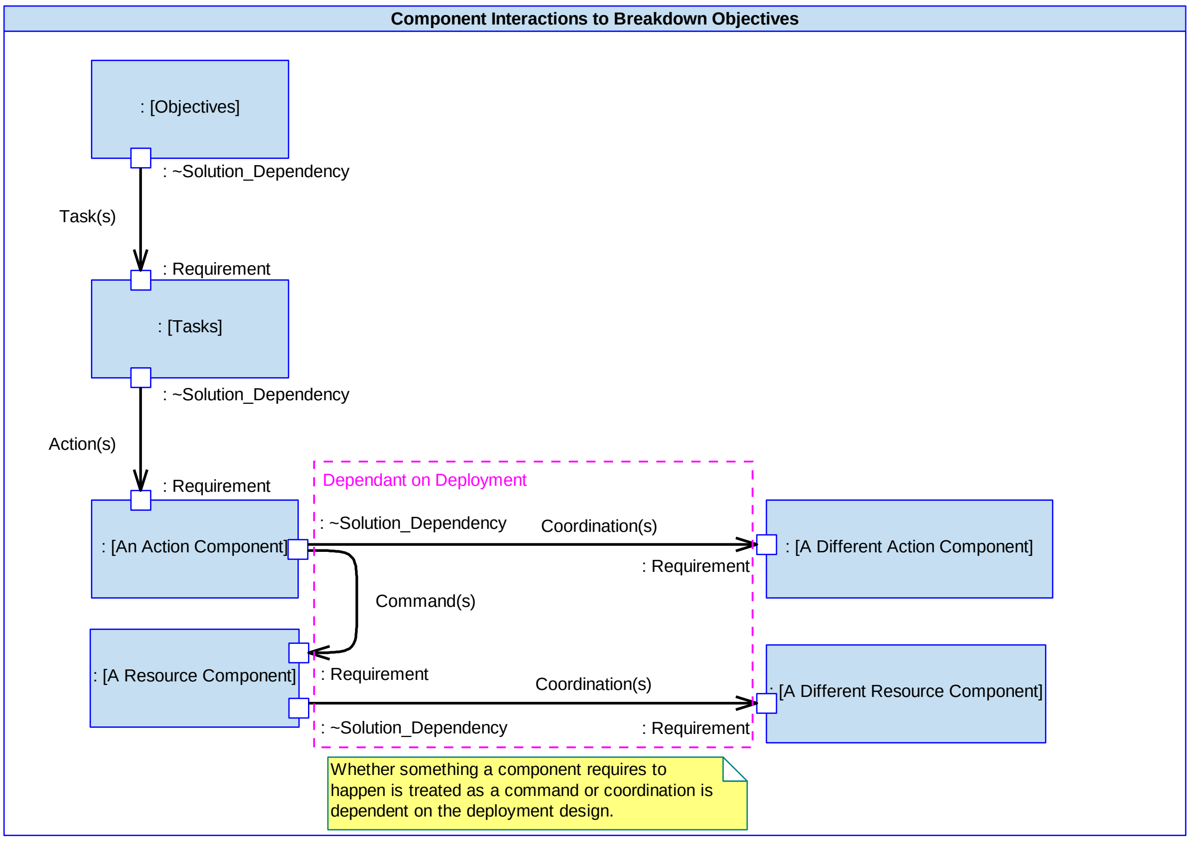


Figure 17: Component Interactions to Breakdown Objectives

The following concepts (covered in detail by other PYRAMID concepts) are also critical to how objectives are suitably broken down by components, and turned into requirements at all levels of abstraction, to allow objectives and requirements to be met in a way that is both operationally viable and supports safe operation:

- Constraint Management - explains how constraints can apply to and be applied by any component in the control architecture and used to limit how an activity (e.g. activities required to satisfy an objective) can be achieved. (See Constraint Management PYRAMID concept.)
- Capability Management - explains how changes in the capability of systems can impact the ways to achieve activities (e.g. activities intended to satisfy an objective). (See Capability Management PYRAMID concept.)
- Resource Management - explains how the allocation of limited resources, required to satisfy requirements, is managed. The Resource Management PYRAMID concept provides a framework that can be adopted to help ensure that limited resources are used in a way that most appropriately satisfies an objective or set of competing objectives.

- Autonomy - explains how different components acting together can provide a system capable of supporting different levels of operator interaction and authorisation associated with tasks, actions, or steps that form part of a solution. (See Autonomy PYRAMID concept.)
- Dependency Management - explains the breakdown of requirements, such as objectives, into derived requirements, such as tasks. It also covers how conflicting requirements and constraints (broken down from the same objective or different objectives) are resolved. (See Dependency Management PYRAMID concept.)

A.2.1.6.2 Operator Interaction

Authorised operators may interact with components from any of the control architecture layers. For a highly autonomous system, interaction will be mostly at the Objective Layer, but where operators have more fine-grained control they will interact directly with the lower layers of the system.

Figure 18: Authorised Operator Interactions shows how an authorised operator may interact with any layer of the control architecture, defining objectives, tasks, actions, steps, or even the use of specific resources as required in the context of a mission. The authorised operator shown on the diagram could be replaced with another system, which itself may handle requirements from an authorised operator. A mission planning tool, for example, could provide a detailed plan that includes the timing of actions and the use of resources. A component that receives requirements from an operator will carry out those requirements by fulfilling its responsibilities in the usual way. It must maintain consistency with other components, negotiate resource use, handle dependencies and use other components as necessary to carry out the requirements. Except for those components that explicitly represent operators as part of their subject matter, components will not explicitly know about an operator but they will need an understanding of the requirements being placed upon them along with abstracted knowledge about their source or origin (i.e. a generic origin ID). This is needed to support prioritisation and the application of rules to resolve any issues with fundamentally conflicting requirements from different sources. It is up to the Exploiting Programme to determine whether a component needs to consider the source of the information that is provided to it.

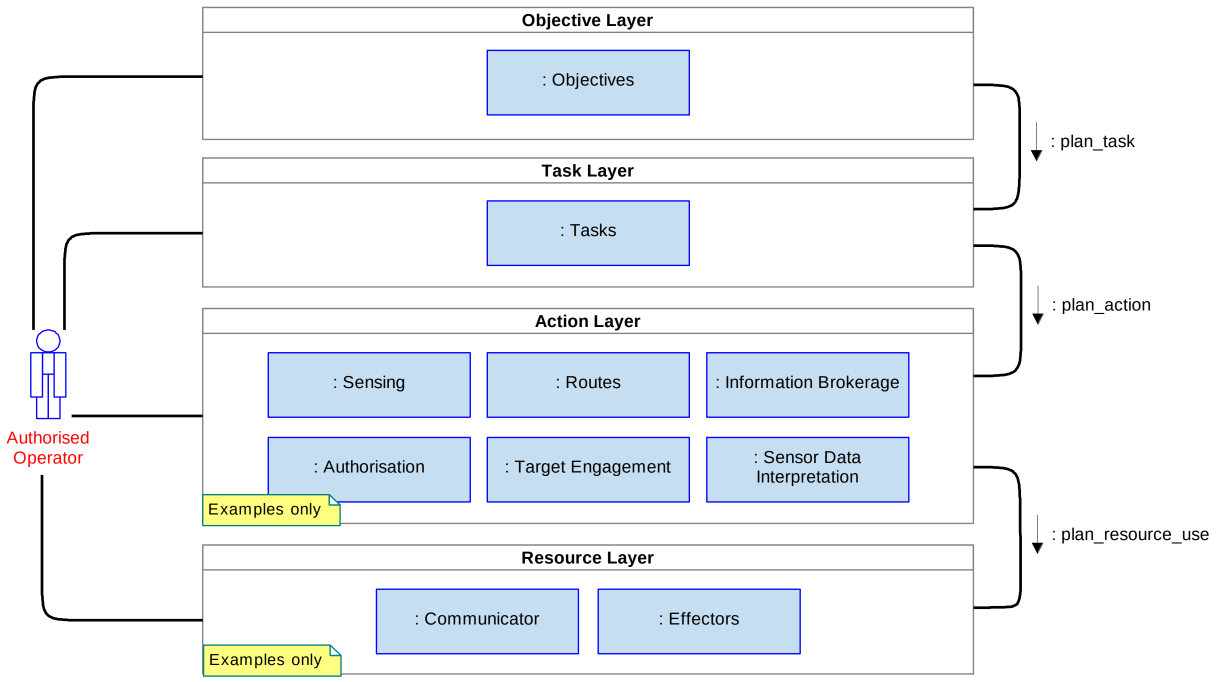


Figure 18: Authorised Operator Interactions

Figure 18: Authorised Operator Interactions shows an abstracted overview of the interactions at each layer, these interactions can occur during any phase of a mission's lifecycle, such as in pre-planning to interactions during mission and post mission debriefs. The authorised operator is able to interact with each layer independently by the direct interaction with components. At which level the operator will interact will vary based on where in the mission lifecycle the interaction takes place. During the execution of the planned mission, for example, the level of interaction could vary as the operator chooses to increase or decrease the amount of autonomy that the system is allowed, see the Autonomy PYRAMID concept. The interaction from the authorised operator can take place at each layer:

- Objective Layer - Setting high level mission goals to be achieved, e.g. the suppression of enemy air defences within a specified region.
- Task Layer - The input of a task important for the completion of an objective, e.g. the destruction of a specific land target to ensure that the overarching objective can be achieved.
- Action Layer - Actioning a component with requirements to create a solution or via the direct input of a predetermined solution, e.g. inputting a set of requirements for a route or a specific route to aviate.
- Resource Layer - Direct control of resources, e.g. selecting the frequency channel for communications equipment.

All PYRAMID deployment interactions between an authorised operator and the system are mediated by HMI components, as described in the Human-Machine Interface PYRAMID concept.

A.2.1.6.3 Resource Component Interactions

The services provided by the Sensors, Effectors and Communicator resource components will likely be utilised within a deployment by other resource components that reason about other equipment resources or physical commodities in order to sense and interact with the physical world, unless interacting with equivalent capability of these components outside of the PYRAMID deployment.

Figure 19: Example of Resource Layer Component Interactions shows a simple example of two of these lowest level Resource Layer components (Effectors and Sensors) that deal with specific equipment interacting with another Resource Layer component, Fluids. In this example the Fluids resource component reasons about the quantities of fuel available in a vehicle's fuel tanks, but utilises services provided by the Sensors component to gather information about the presence of fuel in any specific tank, and utilises services provided by the Effectors component to open and close valves needed to move fuel between tanks.

The software infrastructure (including both middleware and operating system) has been shown as a series of functions that may exist between the components that reason about the lowest level of specific equipment and the actual equipment. For example, taking the control request in data form from Effectors, translating that, for example, into a voltage on discrete lines and outputting that voltage on discrete lines to the effector.

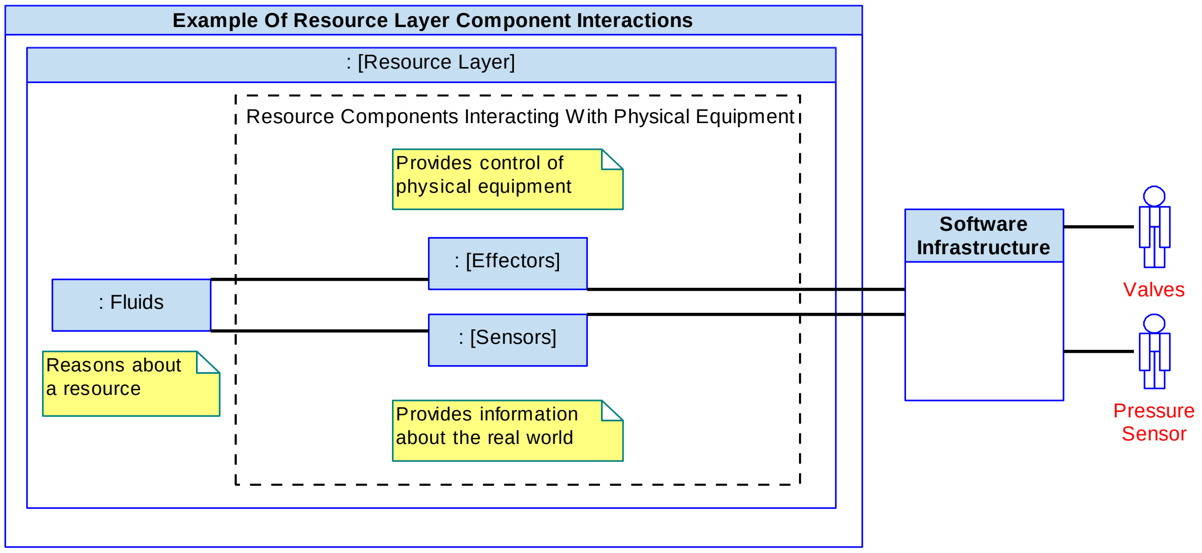

Example Of Resource Layer Component Interactions
: [Resource Layer]
Resource Components Interacting With Physical Equipment
Provides control of physical equipment
: [Effectors]
: Fluids
: [Sensors]
Reasons about a resource
Provides information about the real world
Software Infrastructure
Valves
Pressure Sensor

Figure 19: Example of Resource Layer Component Interactions

A.2.1.7 Control Architecture Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Control Architecture Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept defines an optional (see exploiter consideration below) layered framework for a control architecture, and highlights specifically how it supports the following aspects of a mission system: • Delegation of requirements - Requirements are delegated through a layered framework ensuring problems are resolved at the lowest possible layer (e.g. conflicting requirements) while providing overall control. • System operation at varying levels of autonomy - User or external system requirements can be injected into the layered framework depending upon the level of autonomy needed to be represented. The higher 'layers' support higher levels of autonomy/abstraction, e.g. a deployment with a low level of autonomy is likely to focus on lower 'layers' of the control architecture framework, with simple, or absent, components at the higher layers. • Interactions with external system resources - Capability and resources provided by equipment external to the PYRAMID deployment can be integrated at different levels of abstraction under a PYRAMID deployment and accessed where required in the layered control architecture framework. • Coordinated control across multiple systems/vehicles, enabling integrated or compatible systems being capable of achieving interoperable, coordinated and holistic solutions.	Control is distributed throughout the PRA and therefore there are no central management components; each component is responsible for managing the activities within its subject matter. • All PRA components support requirement handling and delegation. The corresponding component responsibilities and services are described in the Dependency Management PYRAMID concept. • All PRA components are responsible for determining their autonomy levels and understanding when authorisation for their actions is required. The corresponding component responsibilities and services are described in the Autonomy PYRAMID concept. • All PRA components support interaction with an operator, for the purpose of exchanging subject matter information and influencing component behaviour. This is described in the Human-Machine Interface PYRAMID concept. • Interactions with an operator - The authorised operator is able to interact with each layer in the control architecture independently through direct interaction with components. • All PRA components can interact with an external system or standalone equipment where the nature of the interaction is consistent with the PRA component subject matter. This is described in the Interaction with Equipment PYRAMID concept. • All PRA components support coordination between different instances or variants of the same component, or between different components, which span multiple systems or vehicles. Component support for multi-vehicle coordination is described in in the Multi-Vehicle Coordination PYRAMID concept.
Specific PRA Component support for the PYRAMID Concept	The PYRAMID concept defines the following control architecture framework component categories: • Objective Layer - Responsible for the management of mission objectives through the execution of tasks. It receives requirements from outside the system that define the mission objectives and form the mission context. • Task Layer - Responsible for achieving tasks through the coordinated execution of actions. • Action Layer - Responsible for performing actions by managing resources and identifying dependencies. • Resource Layer - Responsible for managing resources in support of the actions that are managed at the Action Layer. These components either reason exclusively about resource equipment or about non-equipment resources (e.g. fuel). • Service Components - Responsible for the provision of information and performing of activities used to support other components.	Each component in the PRA (73 in total) is notionally assigned to a control architecture layer which bounds their scope in achieving mission objectives. • The PRA defines a single component, Objectives, within the Objective Layer. • The PRA defines a single component, Tasks, within the Task Layer. • The PRA defines 25 components in the Action Layer, each with specialised responsibilities and services (from the Component Composition) that reflects the component's role as an Action Layer component and the specific nature of its subject matter. • The PRA defines 10 components in the Resource Layer, each with specialised responsibilities and services (from the Component Composition) that reflects the component's role as a Resource Layer component and the specific nature of its subject matter. • The PRA defines 36 Service components, each with specialised responsibilities and services (from the Component Composition) that reflects the component's role as a Service component and the specific nature of its subject matter.
Exploiter considerations	The development and subsequent implementation of a specific control architecture framework within a deployment is an Exploiting Programme consideration.	The PRA does not mandate the implementation of the PRA defined control architecture approach or the associated component categorisations.

Table 1: Control Architecture PYRAMID Concept Summary

A.2.2 Constraint Management

A.2.2.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Dependency Management

A.2.2.2 Introduction and Scope

This PYRAMID concept explains how the concept of managing system constraints can be applied through the use of the PRA.

A.2.2.2.1 What are Constraints?

In the context of the PRA a constraint is a limitation on the behaviour of a PYRAMID deployment at any level (whole system or constituent part). A constraint is the actual limitation rather than the rule or policy from which that limitation is derived; e.g. a rule might prohibit the use of electronic warfare measures, however the associated constraint will be on the use of a particular piece of equipment. Some of these limitations are fixed, e.g. the aircraft's structural limits, and require no system management. Others are more flexible and need to be managed so that the exact extent of the limitation imposed on system behaviour at a point in time is understood. Thus constraint management can be considered to be the structured imposition or relaxation of limitations based on rules and policies. Note: As far as the PRA is concerned, external rules and external policies are equivalent in that they will both lead to constraints. For clarity, this PYRAMID concept will only refer to rules, therefore a reader considering an external policy such as an Emissions Control (EMCON) policy should apply the concepts identified here for rules to the external policy under consideration.

A constraint should not be considered to simply be any form of negative requirement; where requirements, and their associated solution dependencies, are described in the Dependency Management PYRAMID concept. Generally, requirements exist to elicit the specific functionality of a component or another system, whereas a constraint will limit the behaviour of the constrained system and will therefore limit all components within the system that are capable of violating the constraint. A constraint need not be expressed in the negative; for example, when limiting where an air vehicle is allowed to fly, the constraint could be expressed as either the regions that are allowed to be flown within or those that must not be entered.

Whereas a requirement will generally have feasibility, achievement, and progress reporting associated with it, a constraint is in place and expected to be adhered to at all times whilst enforced. (Note that the constraint may have a context associated with it that defines the circumstances under which it is applicable, such as the time period within which it is enforced.) A constraint will limit the behaviour of the system even though the full capability to surpass that limit may be available; for example, a constraint on the regions that are permitted to fly within, where the air vehicle has the capability to fly outside the limit. If the constraint is not adhered to, due to external factors, unforeseen circumstances, or it being impossible to do so, then the constraint should be reported as being breached by the relevant component(s); for example, if the air vehicle is blown off course resulting in it entering a prohibited air zone. Generally, individual components should do everything that is reasonably practicable within the scope of their responsibilities to try and rectify the breach so it does not continue to occur. A constraint can ultimately result in activities being carried out, which otherwise would not be carried out.

A.2.2.2.2 Types of Constraints

There are many types of constraints, but for the purposes of the PRA, at a system-level they can be categorised into two broad categories:

- **Fixed:** Fixed constraints are those imposed by the physical capabilities of the vehicle. Although a PYRAMID deployment could be configured to make varying levels of allowance for such a constraint the underlying constraint itself cannot be managed by a PYRAMID deployment. Therefore they are considered to be outside of the scope of this PYRAMID concept. Rules adopted to account for fixed constraints are within scope and fall within the category of flexible rule-based constraints outlined below.
- **Flexible:** Flexible constraints are those imposed as a result of some form of rule, policy or variable consideration. These are subject to management by the system based on the rules from which they are derived and so are within the scope of this PYRAMID concept. Flexible constraints can be further subdivided into three sub-categories:
    - **Rule-Based:** Rule-based constraints are derived from a fixed set of rules and apply to all activities; there may be conditions attached to these rules to determine when a rule is applicable, e.g. vehicle location. Examples of rule-based constraints are those derived from rules of engagement or emissions control policies.
    - **Solution-Based:** Solution-based constraints are derived as part of a problem solution; they can only constrain the system within the bounds allowed by the currently applicable rule-based constraints (see Figure 20: Effect of Constraints on PYRAMID Deployment Decision Making). Examples of solution-based constraints are those derived from tactics.
    - **Internal:** Internal constraints are defined internally within components. These can either be hard-coded or data-driven into a component. As with all constraints, internal constraints have contextual information which defines when the associated constraint is applicable.

A.2.2.2.3 Why are Constraints Important?

An activity or activities undertaken in the world are subject to some form of constraint, ranging from the physical capabilities of the person or system undertaking the action, through to legal or moral concerns. A system acting autonomously (on behalf of a human) or semi-autonomously (in co-operation with a human) should act in a manner consistent with the desires of the human who is ultimately responsible for the actions of the system. Equally a system acting in an advisory role should provide advice that is valid and therefore can be accepted: advice which is invalid, and therefore would never be accepted because it breaches rules is worse than no advice at all because it is a wasteful distraction. To achieve these goals the system needs to have its range of actions bounded in some way. For example, the Autonomy PYRAMID concept highlights the use of constraints and interpretation of rules within the Operational Rules and Limits component to control the limits of autonomous decision making. This restricts components from making inappropriate autonomous decisions without the correct authorisation.

A.2.2.3 Overview

The PYRAMID concept for constraint management can be summarised as follows:

- Rule-based constraints are managed in a centralised manner: a single component is responsible for the interpretation of rules and the determination of which constraints the system is consequently subject to.
- Solution-based constraints are managed in a more distributed manner. They form part of a component's solution to a problem, which may be directly in response to satisfying a requirement, a consequence of addressing an imposed constraint, or a combination of these. In the case where a general rule-based constraint has been imposed, a more specific constraint may be derived as needed to support the solution. From the perspective of any one component planning a solution, the component does not know whether a solution-based constraint would be interpreted either as a constraint or a requirement on a different part of the system.
- Internal constraints are defined within a component, either as hard-coded or data-driven data, and so are not explicitly imposed by other components or from outside of the system. Whether or not the constraint is active can be determined from information that the component receives informing it of the current context, for example the constraint may be active whenever the air vehicle is airborne.

A.2.2.4 Component Composition

This PYRAMID concept identifies a number of generic component behaviours in relation to Constraint Management. These behaviours are formally captured as component responsibilities in the PYRAMID Technical Standard, section Component Composition, along with associated component services. These responsibilities and services are also summarised in Table 2: Constraint Management PYRAMID Concept Summary.

Appendix B: Use Cases demonstrates how these responsibilities and services support component interaction in relation to Constraint Management.

Note that these services are also used within the component interaction patterns and examples provided within this concept.

A.2.2.5 Applicability of Rules

The rules with which a system must comply are generally defined using a structured approach which progressively applies more detail at each stage of definition.

- First a 'library' of rules is established; this library generally covers a broad spectrum of users and hence not all rules in the library will be relevant to a particular usage of a system. Nevertheless any rules to which a system is subject would be drawn from the library and it is the library that provides the set of rules which would need to be related to system constraints.

- Next, the rules in force for a particular set of circumstances are defined; these are the ‘currently active’ rules. Examples of these might be the rules of engagement in force in a particular theatre of operations or the EMCON policy in force in a particular geographic area.
- However even the ‘currently active’ rules will have conditions applied to them leading to the final stage; the determination of which rules are ‘currently applicable’ (i.e. the rules applicable to the system in ‘situation now’). An example of this is a rule of engagement which prohibits certain activities when certain conditions are met, thus while a rule may be active when the system is in a certain area the associated constraints will only be applied when the defined conditions are met. Another example is a rule regarding the torque limit on the engines of a twin-engine helicopter. In the event of a loss of an engine the conditions associated with the rule may now permit over-torque of the remaining engine in order to get the helicopter home. This is sometimes referred to as the concept of 'soft limits', constraints which can be relaxed when certain conditions are met.

A.2.2.6 Effects of Constraints on PYRAMID Deployment Decision Making

The same constraint can be in place for both rule and solution-based reasons; from the perspective of the constrained area of the system the reason the constraint is in place is not important. Take the following example rules and tactics:

- **Rules of Engagement (**RoE**):** Rule 130C - Use of electronic warfare measures is permitted (example taken from Sanremo Handbook on Rules of Engagement Ref. [12]).
- **EMCON Policy:** Use of Electronic Countermeasures (ECM) in region 1 is prohibited.
- **Tactic:** While within 50km of potential threat, emissions in the direction of potential threat are prohibited (potential threat located in Threat area).

All of these can lead to a constraint on the use of a wingtip Electronic Countermeasures (ECM) emitter. Figure 20: Effect of Constraints on PYRAMID Deployment Decision Making illustrates how at different points the examples above cause imposition of the same constraint. Within the entire area of the diagram, RoE permits the use of wingtip ECM emitters, however, at point A, the port wingtip emitter is constrained by the EMCON policy while the starboard emitter is constrained by both EMCON and tactics. When the aircraft moves into region 2, at point B, the port wingtip emitter is no longer constrained (as the aircraft is outside region 1 the EMCON policy rule no longer applies) but the starboard emitter continues to be constrained.

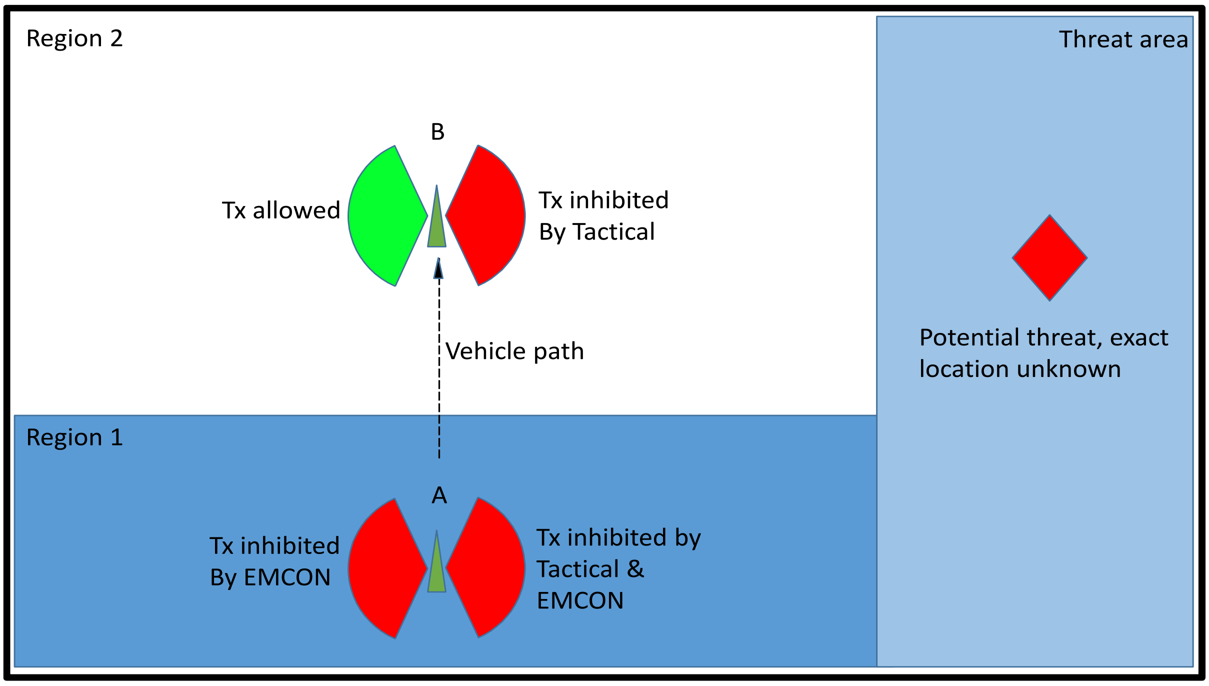


Figure 20: Effect of Constraints on PYRAMID Deployment Decision Making

A.2.2.7 Management of Constraints

The following sections describe the different types of constraint which may be applied to a component.

Note that authorisation is a specific example of contextual information (i.e. the constraint is applicable in any situation where authorisation has not been received), and seeking authorisation as part of a solution is a way of removing the applicability of a constraint.

A.2.2.7.1 Rule-Based Constraints

Rule-based constraints are managed by the Operational Rules and Limits component. This component's function is to understand the rules and the criteria for rule applicability; it will also be aware of the constraints that can be imposed on the system and their relationship to the rules. It will not be aware of the implications of the constraints it determines or what areas of the system are affected by them. The Operational Rules and Limits component will establish which rules are currently active and assess the current conditions to determine if a rule is applicable. If it determines that the criteria for rule applicability have been met it will set the associated constraint. It will also forecast changes to rule applicability.

Restricting knowledge of the rules and their applicability to this one component minimises the number of components exposed to potentially highly classified rules and ensures they are applied uniformly. A limit identified by Operational Rules and Limits is imposed upon another component as a constraint which it must comply with. If events beyond the component's control cause the limit to be breached inadvertently this will result in the component's need to modify the solution and respond appropriately to prevent the breach from continuing to occur. For example, a breach could occur as soon as a constraint is set, if the system is already operating in a way that violates the constraint and time is needed to change the system state, or a breach could occur as a result of unexpected environmental circumstances. The component's constraint service will report the breach; Operational Rules and Limits will then determine which rule has been broken. This rule breach could be reported to an operator or other components to perform remedial actions, or be recorded as a significant event.

A.2.2.7.2 Solution-Based Constraints

Solution-based constraints are managed across the architecture; they are imposed by whichever component develops the ‘problem solution’ which is dependent on that constraint being in place. Solution based constraints may emerge directly in the course of generating a solution to meet a requirement and are identified as constraints that must be observed by components which form part of the solution. As well as resulting from a requirement, solution-based constraints may also result from other constraints imposed on a component. Generally it is expected that solution-based constraints would be imposed by Objectives, Tasks, and the Action layer components (see Dependency Management PYRAMID concept, section Deriving a Solution to a Requirement). Solution-based constraints are managed through a constraint enforcement solution that a component will establish in order to stay within the limits imposed by a constraint. To achieve this, the component may need to generate derived requirements or derived constraints. As with rule-based constraints, it is expected that the components determining the need for a solution-based constraint would have no knowledge of what areas of the system are affected by that constraint. Much in the same way as a rule-based constraint it is also possible for a solution-based constraint to become applicable in a situation where it is immediately breached. This breach will be reported and, where possible, activities should be performed to conform to the constraint.

A.2.2.7.3 Internal Constraints

Constraints may also be defined internally within components. These can either be hard-coded or data-driven. Unlike Rule-Based and Solution-Based constraints, internal constraints are not explicitly imposed by other components. These pre-defined constraints are triggered through consuming information which defines the relevant circumstances or context in which they are applicable, e.g. enacting a constraint on the way that a piece of equipment can be used, due to safety concerns, when information about an equipment failure condition is received. These can also be subject to breaches, and like the other constraints these breaches can be recorded/logged and reported depending on the conditions of the breach. The component will still typically try to rectify the internal constraint breach.

A.2.2.8 Component Composition

This PYRAMID concept identifies a number of generic component behaviours in relation to Constraint Management. These behaviours are formally captured as component responsibilities in the PYRAMID Technical Standard, section Component Composition, along with associated component services. These responsibilities and services are also summarised in Table 2: Constraint Management PYRAMID Concept Summary.

Appendix B: Use Cases demonstrates how these responsibilities and services support component interaction in relation to Constraint Management.

Note that these services are also used within the component interaction patterns and examples provided within this concept.

A.2.2.9 Constraint Examples

This section gives examples of how components might work together when constraints have been placed upon them.

Note that some of the services shown on the components in Figure 21: Rule-Based Example and Figure 22: Solution-Based Example are generic, as defined in the PYRAMID Technical Standard, Ref. [1], section Component Composition. The actual services may have been specialised to the subject matter of the specific components, shown in the PYRAMID Technical Standard, Ref. [1], section Component Set.

A.2.2.9.1 Rule-Based Constraints Example

Figure 21: Rule-Based Example shows an example of rule-based constraints applying to a search task. Operational Rules and Limits has already determined which rules are active. From the active rules, it has also determined the rules that are applicable, and therefore need to be enforced, based on the relevant operating conditions. In order to meet these applicable rules, Operational Rules and Limits identifies the associated limits that need to be adhered to by other components. For example the limits may restrict the use of certain electromagnetic spectrum frequencies due to EMCON rules being enforced within the current operating region.

These limits are interpreted as constraints by the components that they apply to. In the unexpected case that one of these components fails to observe one of the constraints, it reports this (through the Constraints service) and Operational Rules and Limits determines the associated rule that has been breached. This can be reported as necessary, such as to make a user aware of the situation.

Note that, whilst not shown, Operational Rules and Limits can also be requested to identify the limits that would be applicable in a hypothetical scenario rather than the current operating conditions. For example this could help determine if a planned future action will be feasible.

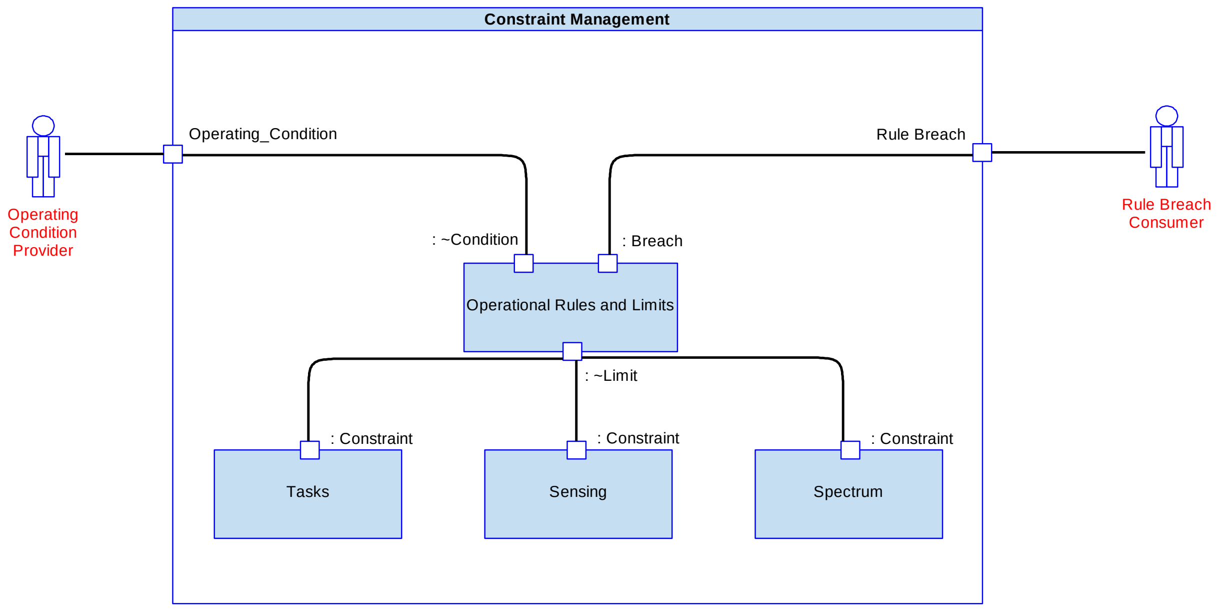


Figure 21: Rule-Based Example

A.2.2.9.2 Solution-Based Constraints Example

Figure 22: Solution-Based Example shows an example of solution-based constraints applying to a search task. The Tasks component determines any solution-based constraints associated with the task. For example, searching should not be carried out in a particular region. Sensing determines solution-based constraints on Sensors associated with the sensing action, such as particular periods of time when the use of the sensor is not permitted. Sensors, taking into account its constraints, determines a solution and places a requirement onto the Sensor Equipment for use during the search task. This example therefore shows how a solution to the search task is determined, taking into account all the solution-based constraints.

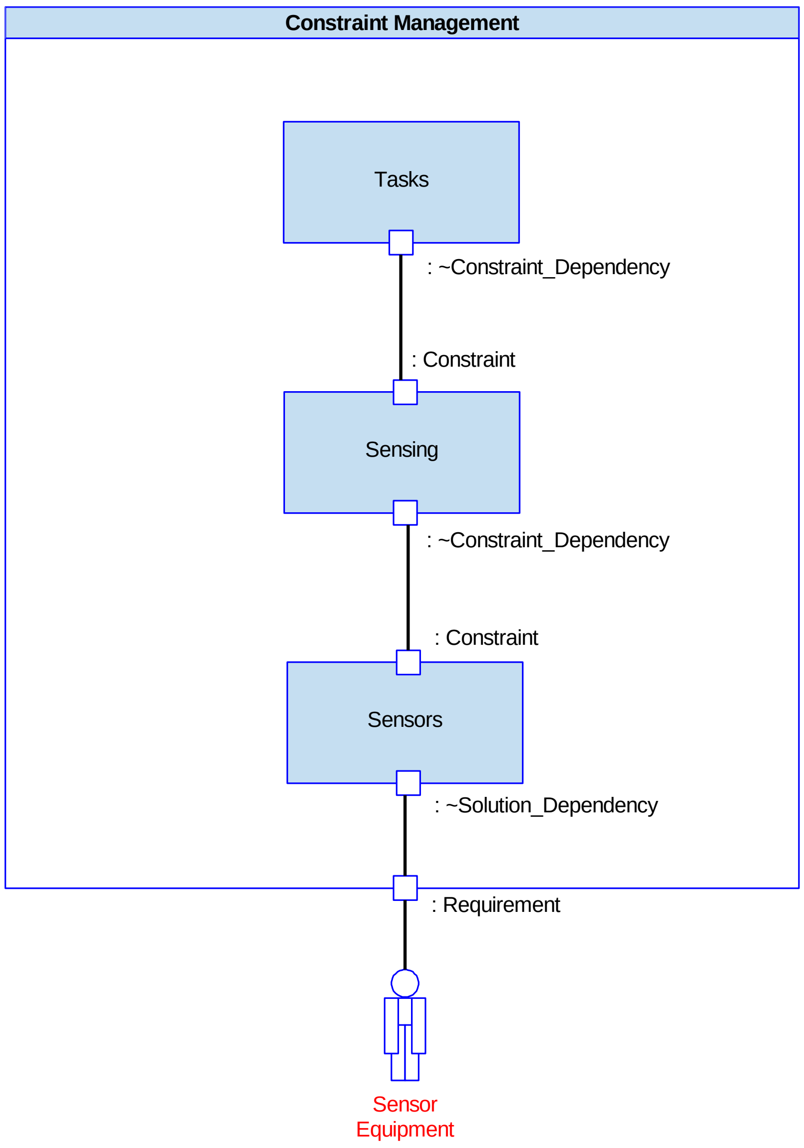


Figure 22: Solution-Based Example

A.2.2.10 Constraint Management Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Constraint Management Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept identifies the following generic component behaviours in relation to constraint management: • Components capture constraints that relate to their subject matter. Constraints may be imposed on a component or internally defined, e.g. through data driving. • Components determine solutions that satisfy requirements taking in to account applicable constraints, or that satisfy constraints directly. • Components identify dependencies when establishing a solution(s) to the requirement placed upon them and may generate derived constraints (to be placed on other components). The policy identifies a category of constraints (Solution-Based) that are derived as part of a problem solution and managed in a distributed manner. • Components identify where constraints conflict with other requirements. • Components may need to seek authorisation for certain activities depending on the applicable constraints. • Components support a process of brokering and arbitration to resolve conflicts. • Components monitor the progress of their own solutions and identify and report where a constraint has been breached.	The PRA defines the following generic component responsibilities and services in relation to constraint management: **Responsibilities:** • capture_constraints • determine_solution • determine_solution_dependencies • identify_conflict • determine_authorisation_dependencies • determine_if_solution_remains_feasible **Services:** • Constraint • Constraint_Dependency For more information on related responsibilities and services please see the Dependency Management and Autonomy PYRAMID concepts.
Specific PRA Component support for the PYRAMID Concept	The PYRAMID concept identifies a category of constraints (Rule-Based) which are derived from a fixed set of rules or policies and that are centrally managed. The PRA separates out the following area of functionality with respect to the management of constraints in this category: • The status and applicability of operational rules and how to derive limits from them.	The PRA defines the following component whose role relates to constraint management: • Operational Rules and Limits
Exploiter considerations	The following constraint management related considerations are a matter for individual Exploiting Programmes: • The method of defining internal constraints within a component (e.g. the use of data driving). • The approach to handling the consequences of constraint breaches.	The PRA does not mandate a specific approach in respect of these considerations.

Table 2: Constraint Management PYRAMID Concept Summary

A.2.3 Dependency Management

A.2.3.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

- Control Architecture

Read in conjunction with

- Constraint Management
- Resource Management
- Component Connections

A.2.3.2 Introduction and Scope

Components can be allocated requirements through their services. When a component cannot fulfil the requirements that are assigned to it directly, it may need to request assistance from other components. This assistance is requested by generating a dependency. This PYRAMID concept describes how these dependencies are managed.

Through these dependencies, components can work together to fulfil the requirements that are placed on the system, along with any derived requirements that are generated internally. When the requirements cannot be met, mechanisms are provided to handle the shortfall and report back to the source.

To support cases where a conflict exists in the requirements that a component is given, which the component is unable to resolve itself, a standard approach to resolving the conflict at a system level is presented.

Behavioural dependencies between components can be expressed in a component interaction pattern. Component interaction patterns are described in the Component Connections PYRAMID concept.

A.2.3.2.1 What is Dependency Management?

PYRAMID components cooperate to achieve the requirements imposed on the deployed system. A major element of this cooperation is to manage the dependencies between components. These dependencies may have different properties:

- **Rigid Dependencies**

  There may be rigid dependencies between components created as part of executing a solution, such as a specific order of actions (e.g. transit to the search area before starting the search). Such dependencies can be implemented by setting up coordination points between actions.

- **Dynamic Dependencies**

  Components may also have dependencies that need to be managed more dynamically. For example, when sensor data is processed it may be discovered that the data was of insufficient quality to fulfil the requirements of the data processing activity being undertaken.

Consequently, it may be necessary to use different sensing techniques, or to fine-tune vehicle manoeuvres, to obtain better data. This kind of dependency can be implemented by establishing a link between cooperating components. As long as the requirements and constraints of the original requirement are not violated, these links can be maintained until the requirement is achieved.

Rigid and dynamic dependencies between components require traceability to facilitate resource management and conflict resolution. See the General Conflict Resolution Strategy section of this PYRAMID concept and Resource Management PYRAMID concept, section Demand Traceability. Rigid and dynamic dependencies are illustrated in the examples in this PYRAMID concept, see Figure 39: Setting up a Sensing Task and Figure 40: Carrying out a Sensing Task.

A.2.3.2.2 Why is Dependency Management Important?

The identification and management of dependencies enables components to cooperate to achieve the requirements placed on them. It is important that dependencies are managed in an explicit and flexible way if a deployment is to be able to adapt to changes in its capability and to external events.

A.2.3.3 Overview

The dependency management PYRAMID concept can be summarised as follows:

- Components identify their dependencies when establishing a solution to the requirement placed upon them.
- To address these dependencies components will generate derived requirements which must be satisfied in order for them to achieve their solution.
- Components monitor the progress of their own solution and determine whether it remains feasible to achieve the requirements it was devised to meet.
- Components report their achievement and progress against their assigned requirements.
- Performance analysis can be used to monitor and improve the quality of solutions as they are being executed.

This PYRAMID concept requires behaviour and services provided by the Component Composition pattern. The sections below (with the exception of Planning Context and Performance Analysis) show how the pattern supports the management of dependencies within the system.

A.2.3.4 Component Composition

This PYRAMID concept identifies a number of generic component behaviours in relation to Dependency Management. These behaviours are formally captured as component responsibilities in the PYRAMID Technical Standard, section Component Composition, along with associated component services. These responsibilities and services are also summarised in Table 3: Dependency Management PYRAMID Concept Summary.

Appendix B: Use Cases demonstrates how these responsibilities and services support component interaction in relation to Dependency Management.

Note that these services are also used within the component interaction patterns and examples provided within this concept.

A.2.3.5 Deriving Solutions

When a requirement is assigned to a component, the component needs to determine a solution that can satisfy it. Each solution includes the elements that the component can satisfy itself, along with solution dependencies that need to be fulfilled by other components. The requirements for these solution dependencies are assessed by other components which determine whether or not they can be achieved.

By managing this chain of dependencies between the components, a complete solution can be generated in response to the originating system requirement. In analysing solutions to multiple system requirements, solutions may also need to consider the interdependencies between those requirements, and the compatibility of solutions for differing requirements.

Whilst this PYRAMID concept generally describes the derivation of solutions in terms of requirements, many of the concepts are also applicable to constraints that require a solution to be enacted (see the Constraint Management PYRAMID concept which describes constraints in further detail).

A.2.3.5.1 Deriving a Solution to a Requirement

When a component has been tasked to provide a solution to a requirement, multiple solutions may be available. The selection of a preferred solution may be done at the component level where the component assesses the solutions to achieve a particular optimisation goal (e.g. reduce resource use) and to avoid any conflicts. Selecting a solution may also be done at a system level which involves a set of solutions across multiple components where there may be trade-offs in selecting one or another set. This more complex situation is discussed in the Overall Solutions section. If selecting a solution at the component level, this selection choice is communicated to the requirement provider and dependent solutions are agreed with subordinate components. Once a solution is agreed, resource allocation can be confirmed and the selected solution is available for execution. Figure 23: Component Solution Derivation shows a simple example where two solutions are considered.

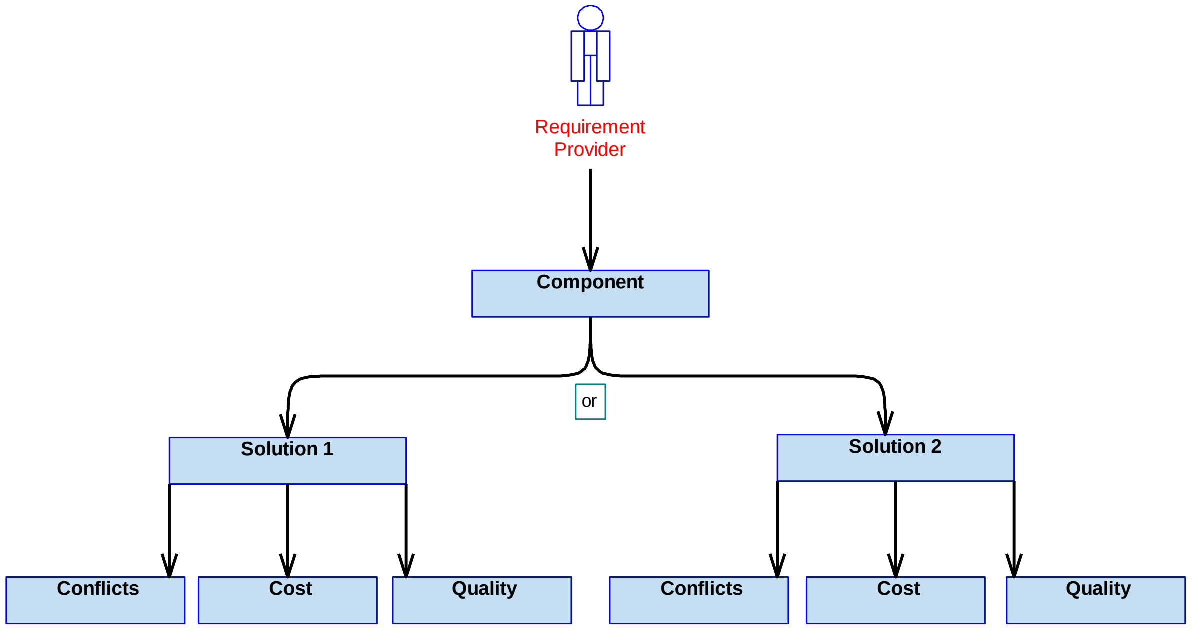


Figure 23: Component Solution Derivation

A.2.3.5.1.1 Pattern Overview

The Determine Solution use case (see Appendix B: Use Cases) shows how a component derives a solution to a requirement. This service pattern uses a three phase exchange, where a requirement is placed, expected achievability is returned, and then if the requirement can be achieved the requesting component triggers enactment of the solution.

Since a component only knows how to satisfy requirements within its subject matter area, components must act together to find available solutions. Two services enable this cooperation:

- The Requirement service allows a component to accept incoming requirements.
- The Solution_Dependency service allows a component to place derived requirements.

In this way, a high level request is broken down through the system to produce lower level requirements (see Control Architecture PYRAMID concept, section Figure 17: Component Interactions to Breakdown Objectives).

Whilst this PYRAMID concept primarily focuses on the use of requirements (and therefore the Requirement and Solution_Dependency services), it is also applicable to constraints (and therefore the Constraint and Constraint_Dependency services) as described in the Constraint Management PYRAMID concept.

A.2.3.5.1.2 Use of the Pattern

The Figure 24: Solution Derivation Example shows a simple example where a Requirement Provider places a new requirement on an Assigned Component. The Assigned Component uses two Contributing Components to achieve the solution.

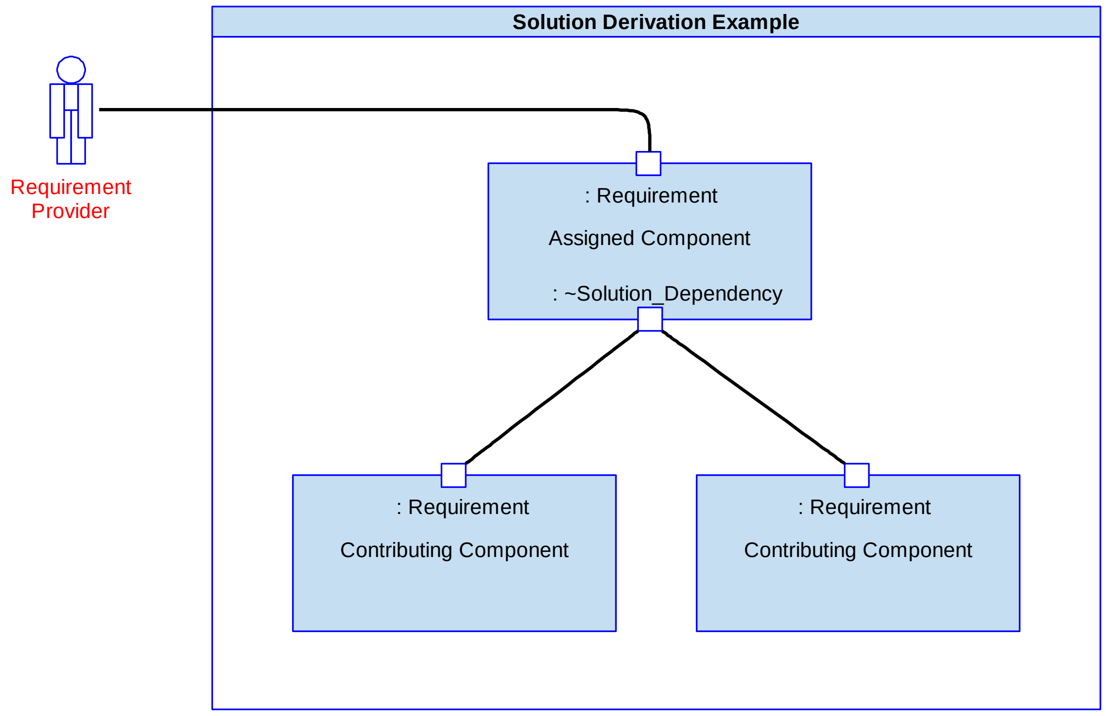


Figure 24: Solution Derivation Example

The Figure 25: Solution Derivation and Execution activity diagram shows the behaviour associated with the example. The Assigned Component receives a requirement. It determines a suitable solution that includes solution dependencies, which it places on the Contributing Components. The Assigned Component is responsible for ensuring that the solution dependencies can be satisfied while the solution is being executed. If the solution dependencies cannot be satisfied, the Assigned Component will seek to adjust the solution to continue to satisfy the requirements (as covered by the Feasibility Assessment section).

Where dependencies change, the Assigned Component may need to adjust the requirements it places on the Contributing Components.

In order to comply with the operational rules of the system, it may be necessary to seek authorisation (e.g. from an external system or authorised operator) to proceed with a given activity. This could be managed by including a dependency on the Authorisation component. Any activities that require authorisation are prevented from being carried out until appropriate authorisation has been obtained.

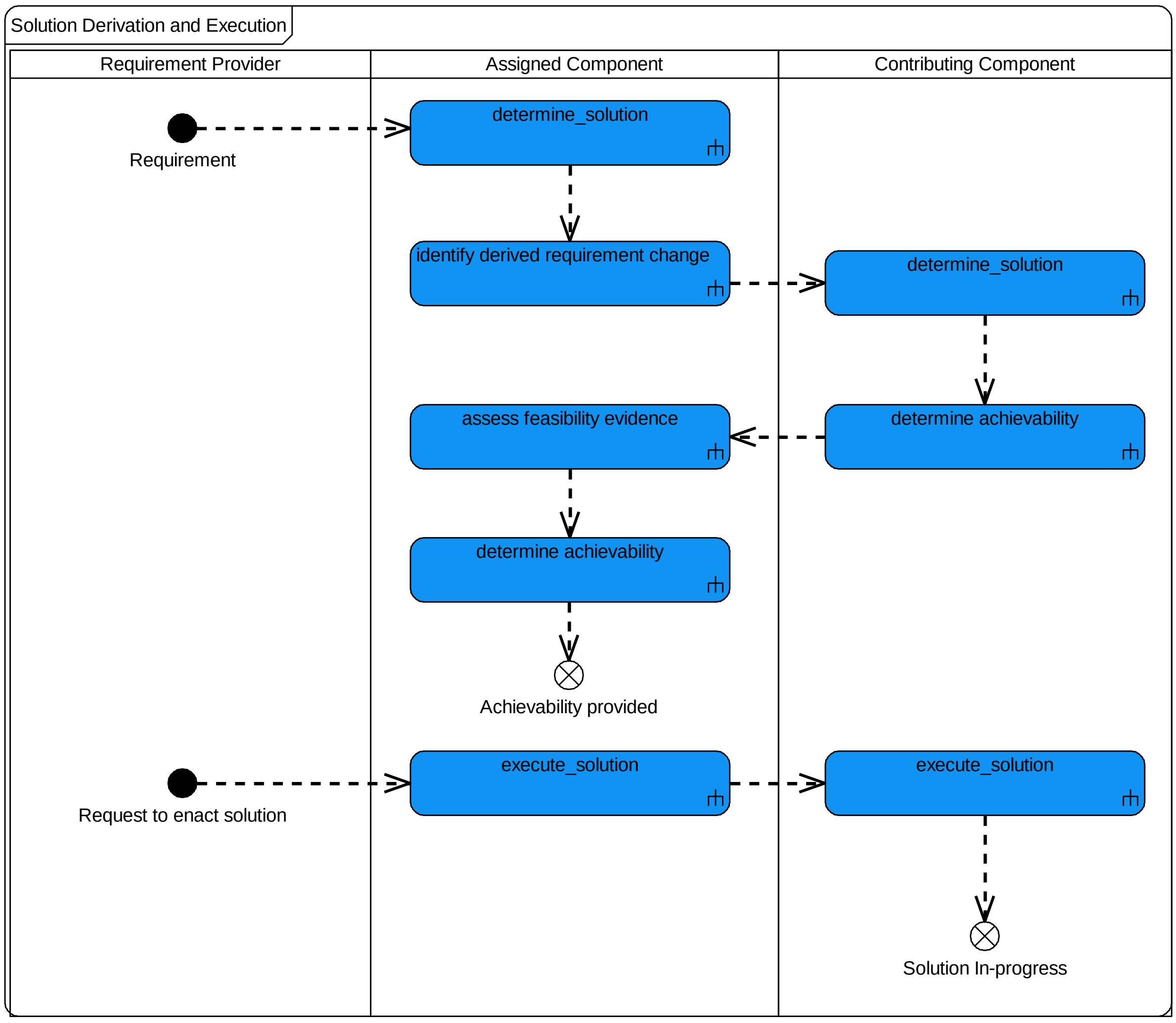


Figure 25: Solution Derivation and Execution

A.2.3.5.2 Deriving Solutions to Multiple Requirements

The previous section discusses how a single component responds to a single requirement placed on it. In a deployment a component may have to respond to multiple requirements and constraints. A component must therefore be able to understand which requirements and constraints are related and need to be addressed collectively. This is necessary to ensure the coherence of any given solution, for example that there are no internal conflicts (any identified conflicts having been resolved), and to distinguish unrelated planning elements which can be treated independently.

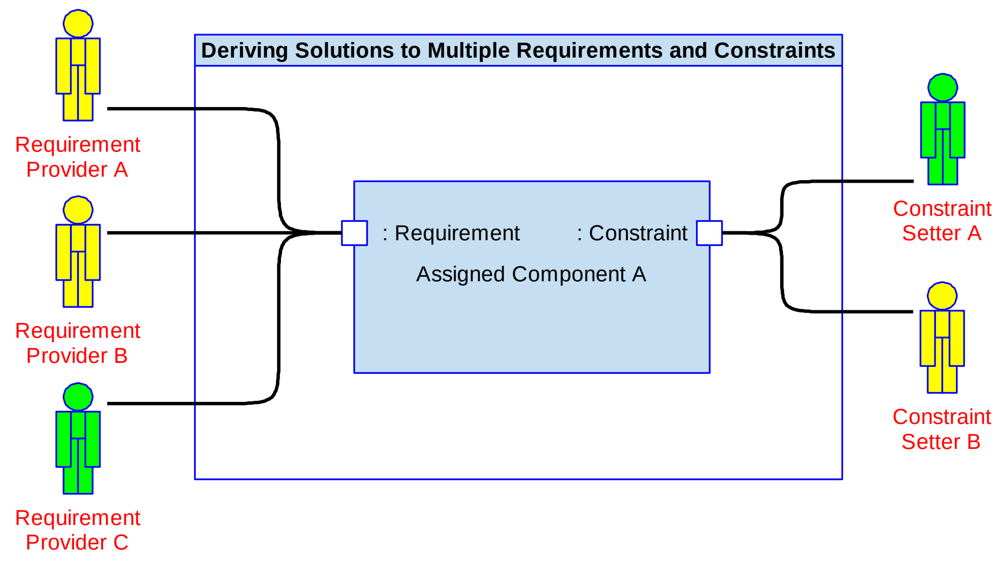


Figure 26: Deriving Solutions to Multiple Requirements and Constraints

Figure 26: Deriving Solutions to Multiple Requirements and Constraints shows a situation in which a component receives multiple requirements and constraints; the diagram uses yellow and green colour coding to differentiate the related sets (although not shown, requirements from the same provider do not need to be bound together and can be separated into different sets). This distinction is essential in understanding that two separate solutions may be needed, which are unrelated, and that the component can treat them independently. Any given solution can then be handled as seen in the previous section, Deriving a Solution to a Requirement. The PRA does not define any mechanism to enable components to identify related and unrelated sets as this could be achieved in a number of different ways with the most appropriate approach depending on the deployment details.

A.2.3.5.3 Overall Solutions

Each component only has visibility and understanding of its own solutions. A system level requirement may be met by a combination of solutions across multiple components, this is what is meant by an overall solution. I.e. the requirement is achieved by executing solutions across a set of components. Additionally, there may be multiple alternate overall solutions possible using components differently or using different component entirely.

By generating alternative solutions, a system is able to:

- Develop candidate solutions which can be tested for viability.
- Provide multiple viable options which can be compared for quality and performance.
- Prioritise or optimise different specific aspects of the solution, e.g. to reduce resource use or improve an aspect of performance.
- Allow re-planning of a solution when circumstances change, such as when the original solution is no longer achievable.

The management of alternative solutions introduces potential complexity; as multiple components could generate multiple solution options. These alternative solutions may each generate their own differing dependent requirements and constraints. Where this occurs, a complex set of interactions may emerge from which a preferred overall solution needs to be discerned.

Furthermore, there can be various stages in the lifecycle of solution development and execution, such as:

- Planning a possible solution to be considered for selection.
- Resolving conflicts.
- Selecting a preferred solution for future execution.
- Enacting a selected solution.
- For audit purposes, retaining a solution that has completed its execution (regardless of whether it was successful or not).

In the next section the concept of planning solution spaces and planning contexts is introduced, which can be used to help manage and mitigate this complexity.

A.2.3.5.3.1 Planning Solution Spaces

As described above, a solution to a system requirement typically consists of a coherent set of individual component solutions, each addressing a specific aspect of the overall problem, according to their subject matter. By coherent, we mean that the relationships and dependencies between these individual component solutions are understood and have been taken in to account, ensuring that there are no conflicts. This may include, for example, dependent requirements and constraints.

Where a component develops alternative solutions that satisfy a requirement, the alternatives may result in different sets of relationships and dependencies. Thus, at a system level, we now have multiple sets of individual component solutions that can be considered coherent. We refer to each of these sets as a planning solution space. Equally, if there are multiple unrelated system requirements, that can be addressed independently, the solutions to these can be considered as within separate solution spaces. Figure 27: System Solution Spaces shows the sets of requirements and constraints from the Figure 26: Deriving Solutions to Multiple Requirements and Constraints example within the context of different solution spaces.

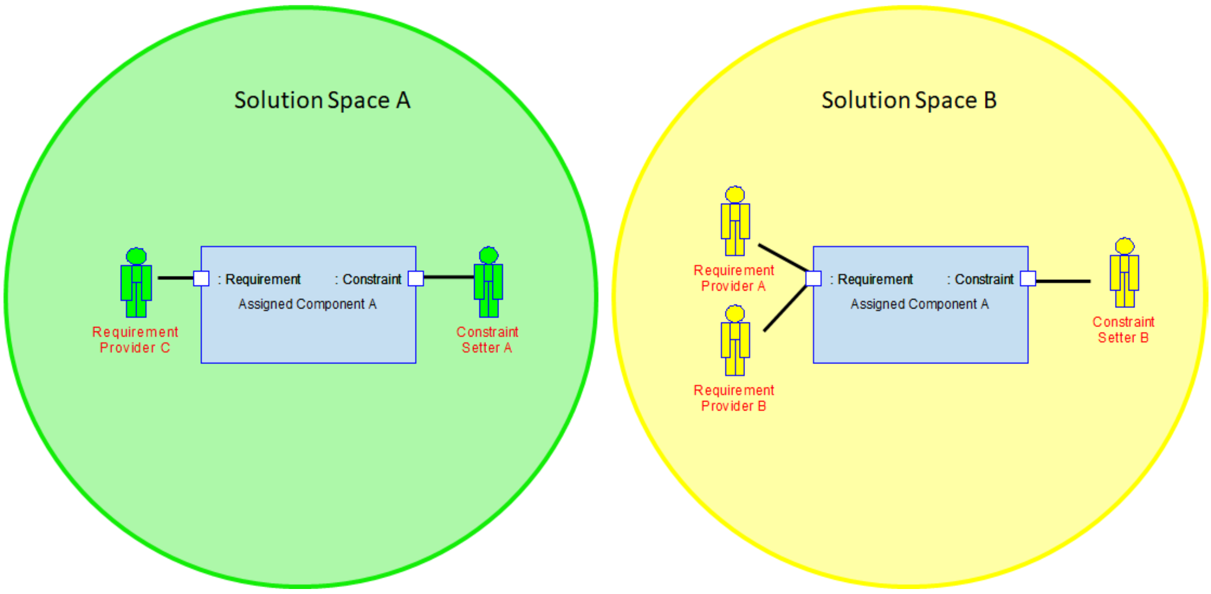


Figure 27: System Solution Spaces

Any individual component solution therefore exists within the context of a specific solution space, i.e. it is consistent with the other component solutions within that space, and the set of relationships and dependencies specific to that solution space are understood and taken in to account.

Where multiple components develop alternative solutions to satisfy their requirements, the number of solution spaces increases as seen in Figure 28: Solution Spaces during System Planning. The number of solution spaces requires careful management. On the one hand, the development of solution spaces is important to enable the optimisation of solutions at a system level by allowing trade-offs to be identified and selected. On the other hand, the number of solution spaces needs to be kept to a manageable level recognising where optimisation choices can be made at a local level.

Any alternative planning solution developed within a single component (within the scope of its subject matter) results in a new, alternative planning solution space for the whole system.

This does not mean that every component involved in the overall solution will be impacted by the new solution space; however, there is potential for any component involved to be impacted.

This can be seen in Figure 28: Solution Spaces during System Planning. It should generally be possible to execute existing planning solutions at a future point in time, without creating a new solution space, if there are no requirement or context changes (other than the point of execution).

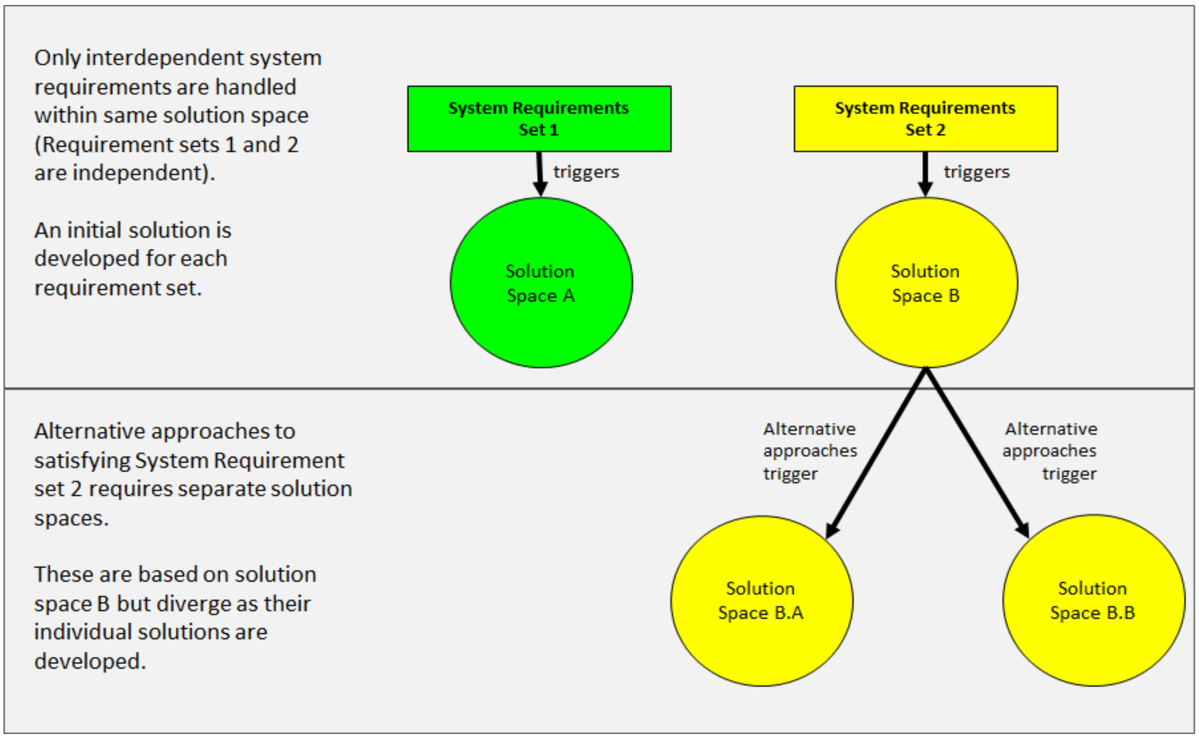

Only interdependent system requirements are handled within same solution space (Requirement sets 1 and 2 are independent).
An initial solution is developed for each requirement set.
System Requirements Set 1
System Requirements Set 2
triggers
triggers
Solution Space A
Solution Space B
Alternative approaches to satisfying System Requirement set 2 requires separate solution spaces.
These are based on solution space B but diverge as their individual solutions are developed.
Alternative approaches trigger
Alternative approaches trigger
Solution Space B.A
Solution Space B.B

Figure 28: Solution Spaces during System Planning

While each system requirement may result in a unique solution space, if there is an interdependency between the different system requirements then they should be considered to exist in the same solution space. For example, if two system requirements need to be satisfied simultaneously, or completed in a mandated sequence, then they will share a common solution space.

Where system requirements do not have interdependencies, separating solutions into separate solution spaces has advantages, including reducing complexity and supporting increased optimisation possibilities. This is particularly true for solutions for different sorties or solutions based on fundamentally different assumptions about the operating context and/or environment. This can be seen in Figure 28: Solution Spaces during System Planning, where Solution Space A satisfies system requirement 1 wholly independently of Solution Space B in its satisfaction of system requirement 2.

While Solution Spaces A and B are not interdependent, and can be developed in isolation of each other, it does not exclude the possibility that these solution spaces are compatible, i.e. the achievement of the requirements satisfied within solution space B does not preclude the achievement of the requirement satisfied within solution space A. Where a solution space does not affect the achievement of another solution space, both solution spaces can be achieved in parallel.

It is possible that there may need to be options for planning to satisfy system requirements both independently or non-independently. For example, it may not be known ahead of time whether the solutions to both requirements will be executed or not; however, if they are then the solution to one will influence what can be achieved to satisfy the other. In this case the user may decide that it is better to achieve just one of the requirements with higher performance than it is to achieve both with lower performance. However, for the user to make an informed choice, a variety of permutations may need to be planned. This is shown in Figure 29: System Requirement Compatible Solution Spaces.

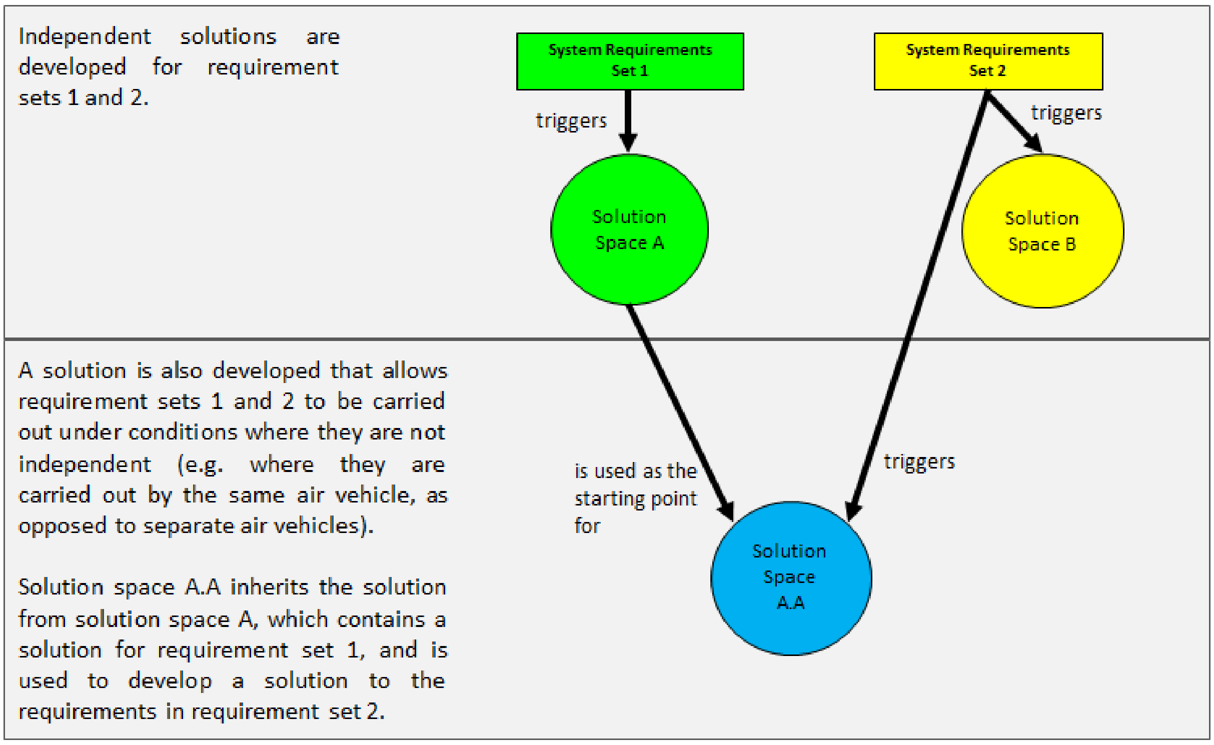

Independent solutions are developed for requirement sets 1 and 2.
System Requirements Set 1
System Requirements Set 2
triggers
triggers
Solution Space A
Solution Space B
A solution is also developed that allows requirement sets 1 and 2 to be carried out under conditions where they are not independent (e.g. where they are carried out by the same air vehicle, as opposed to separate air vehicles).
Solution space A.A inherits the solution from solution space A, which contains a solution for requirement set 1, and is used to develop a solution to the requirements in requirement set 2.
is used as the starting point for
triggers
Solution Space A.A

Figure 29: System Requirement Compatible Solution Spaces

The diagram shows a particular example, in which the requirements are addressed in a particular order with Solution Space A being used as the starting point for accommodating the additional Requirement Set 2. Alternative, but equally valid approaches include:

- Solution Space B being used as the starting point to subsequently accommodate Requirement Set 1.
- Solution Space C (not shown) being generated to address Requirement Sets 1 and 2 collectively.

This highlights the potential need for the new system requirement to be planned both in isolation from an existing system requirement (i.e. in its own planning solution space) and in conjunction with the original system requirement (i.e. in a common solution space).

The Exploiting Programme will therefore have to choose an appropriate approach that balances design complexity with flexibility and operational effectiveness.

A.2.3.5.3.2 Planning Context

Any planning solution is developed within a context that defines the conditions within which the solution is developed and that the solution must take in to account. For example, such a context may include information regarding the overall operation, the mission, the platform or the environment. When used for planning purposes this contextual information forms part of the planning context. The other key parts to planning context are shown on the Figure 30: Planning Context Entity Relationship Diagram below; it shows the following:

- How an aspect of a requirement defines context within which a solution to the Mission Goal Requirement must be determined. The context is common to all of the solution spaces that aim to address the Mission Goal Requirement.
- How additional contextual information is established during the development of a solution (i.e. the individual Component Solutions) that exists within the Solution Space. This additional contextual information therefore also applies to components developing solutions within that space.
- How a component develops a solution (to meet its derived requirements) within the context of a particular solution space and is therefore bound by the aspects of both Common Contextual Information and Solution Specific Contextual Information that are applicable to its subject matter.

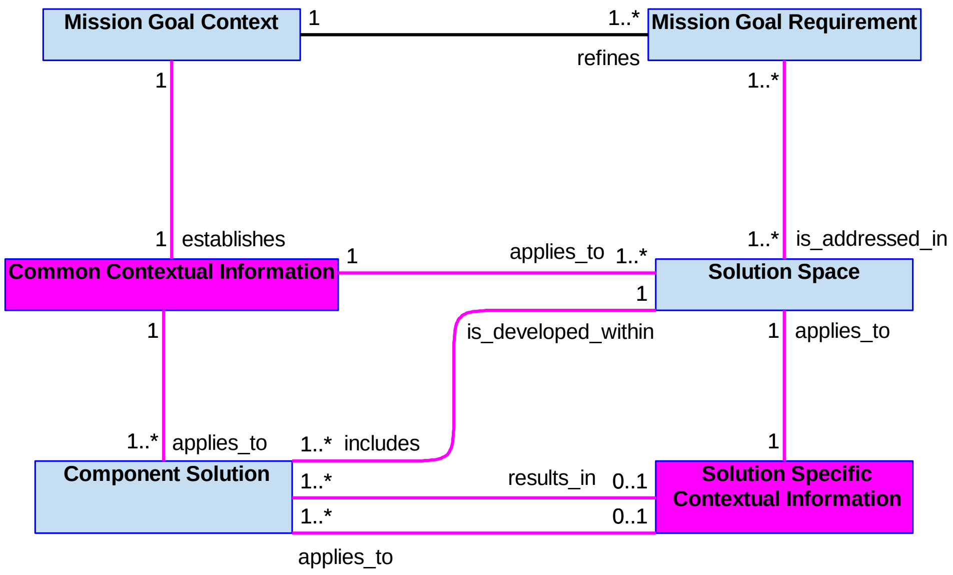


Figure 30: Planning Context Entity Relationship Diagram

Mission Goal Context - A requirement on the system that establishes context for the mission, e.g. the time and location of the mission.

Mission Goal Requirement - A requirement on the system to achieve a particular goal, e.g. destroy a target.

Common Contextual Information - Contextual information that applies to all candidate solutions, e.g. the time window and information about the operating theatre of the mission.

Solution Specific Contextual Information - Contextual information that is applicable to the overall solution being developed within the solution space.

Component Solution - A component's solution to one or more requirements.

Solution Space - A conceptual space that component solutions are developed within.

The planning context can thus be considered from a system perspective and an individual component perspective:

- From a system perspective, when a Mission Goal Requirement is placed on the system, that requirement may be refined by a Mission Goal Context requirement. This Mission Goal Context requirement contains contextual information that defines the circumstances and the conditions within which the solutions to the Mission Goal Requirements are sought. This is Common Contextual Information that applies to all candidate solutions considered when attempting to satisfy any requirement applicable to the mission. For example, this could be the time window in which the mission should be carried out and information about the theatre of operation for the mission that is being carried out.
- From a component perspective, when a requirement is placed on the component, there is also associated context to be taken in to account. Some aspects of the Common Contextual Information may be applicable to this component and form part of its context. However, the context for the component is defined not just by the relevant common context, but by the Solution Space within which the component is working to find a solution, i.e. Solution Specific Contextual Information. For example, the route associated with a particular solution option might define the context within which a communications solution is sought.

An understanding of planning context must be maintained at both levels, and any solution identified is only applicable to the planning context within which it was developed. For any requirement placed on the system it is desirable to determine the applicable contextual information, otherwise sufficiently broad assumptions about the possible context need to be made, covering the full range of scenarios that the system design needs to cater for. However, the system design may still need to cater for situations where the relevant contextual information is not available to the system. This information will be captured in, and distributed across, the PYRAMID components according to their subject matter. For a requirement placed on a component, the component must be able to identify related requirements, constraints and information, including contextual information, i.e. it must be able to identify what is applicable to the solution space within which it is working. Components require the appropriate services to access the contextual information relevant to their subject matter, and to provide information within their subject matter that can be translated into contextual information used by other components. Figure 31: Planning Context Contextual Information depicts a component Information_Dependency service used to access both common contextual information, applicable to all solution spaces, and solution specific contextual information that is applicable to a specific solution space.

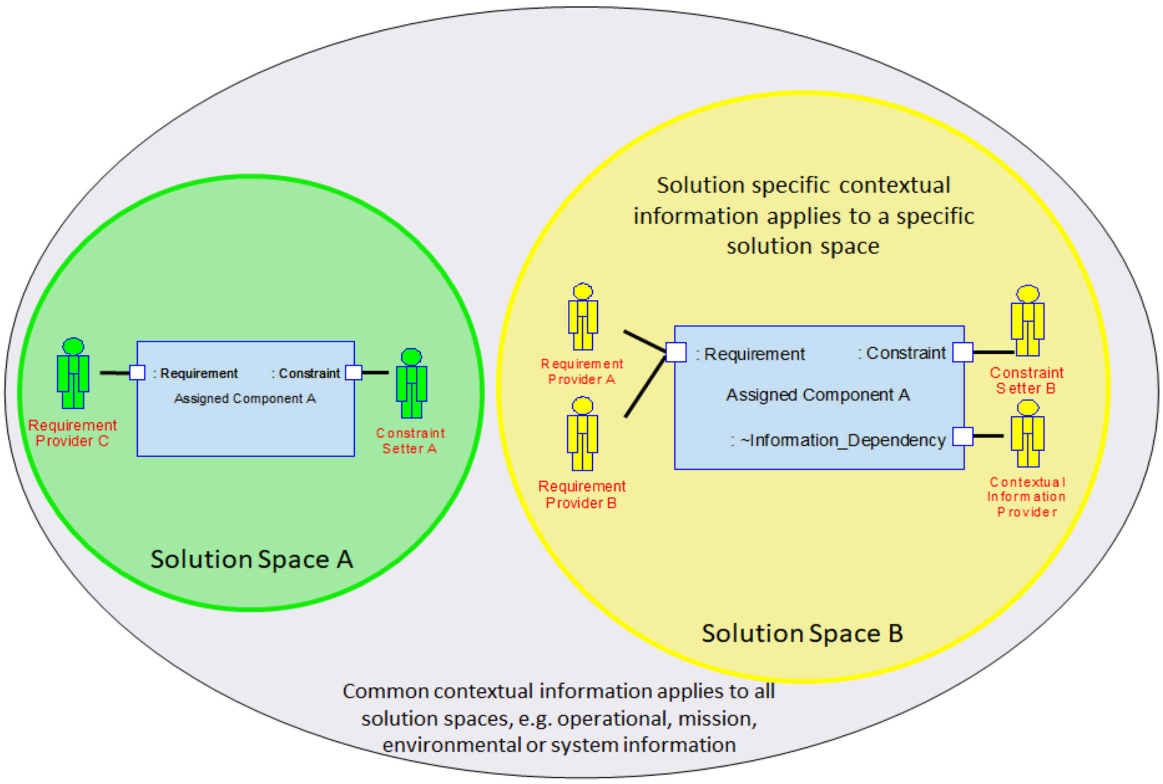

Solution specific contextual information applies to a specific solution space
Requirement Provider C
: Requirement
: Constraint
Assigned Component A
Constraint Setter A
Solution Space A
Requirement Provider A
Requirement Provider B
: Requirement
: Constraint
Assigned Component A
: ~Information_Dependency
Constraint Setter B
Contextual Information Provider
Solution Space B
Common contextual information applies to all solution spaces, e.g. operational, mission, environmental or system information

Figure 31: Planning Context Contextual Information

In general, a component must be able to distinguish related sets of requirements, constraints and information, which must be treated collectively, and unrelated sets, which can be treated independently. This is essential to ensuring coherent solutions, including the avoidance or resolution of conflicts where these occur, as described in the Conflict Resolution section. As also noted above, the PRA does not mandate any particular approach or mechanism by which traceability of such information (e.g. to a common solution space) might be managed, since there are a number of valid approaches and the most appropriate approach could be different for different Exploiting Programmes.

A.2.3.5.4 Conflict Resolution

A.2.3.5.4.1 General Conflict Resolution Strategy

Where multiple demands (e.g. requirements or constraints) are placed on a component and need to be collectively satisfied, conflicts may arise. Figure 32: General Conflict Resolution Pattern shows the generalised pattern for dealing with such conflicting demands. This pattern depicts the typical nature of the component interactions as conflicting demands are identified and subsequently resolved through a process of brokering and arbitration.

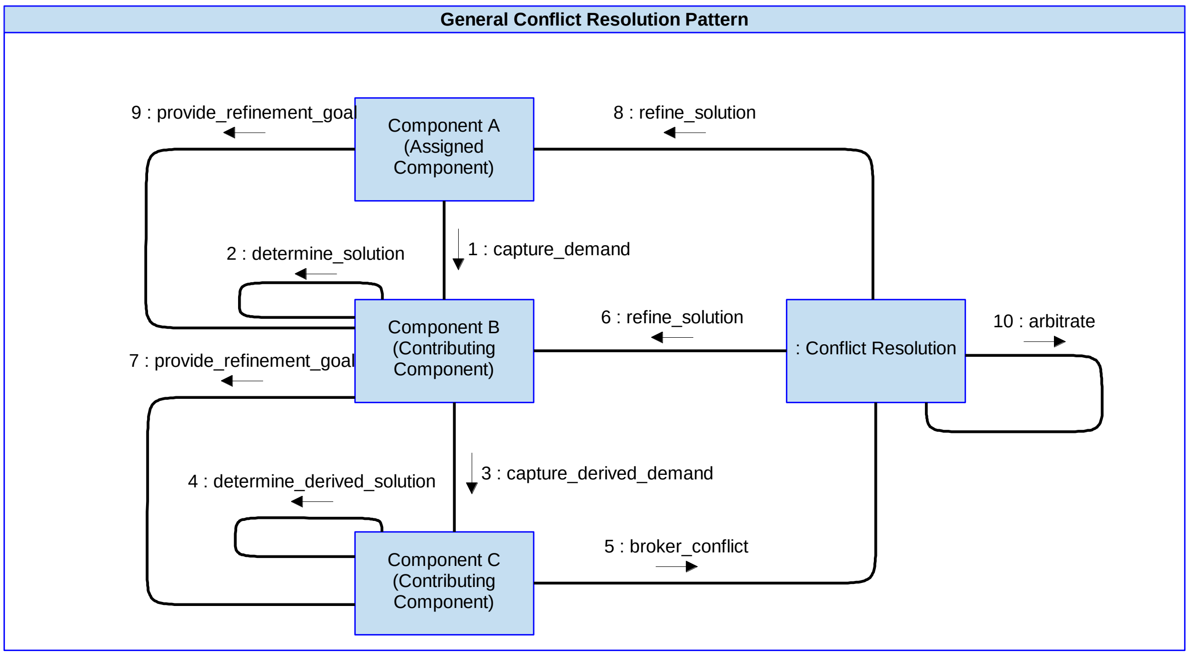

General Conflict Resolution Pattern
Component A (Assigned Component)
Component B (Contributing Component)
Component C (Contributing Component)
: Conflict Resolution
1 : capture_demand
2 : determine_solution
3 : capture_derived_demand
4 : determine_derived_solution
5 : broker_conflict
6 : refine_solution
7 : provide_refinement_goal
8 : refine_solution
9 : provide_refinement_goal
10 : arbitrate

Figure 32: General Conflict Resolution Pattern

The typical flow of events may include the following stages:

Conflict Identification: Component A places a demand on to Component B. Component B determines a solution to satisfy this demand and places a resulting derived demand on to Component C. These demands may take the form of requirements or constraints. Component C determines that the demand placed on it can be met by when considered in isolation but cannot be met in conjunction with additional demands that have been placed by other components (not shown). Since no solution can be found to satisfy all demands, a conflict is identified by Component C and notification is provided to the Conflict Resolution component. The generic Broker_Conflict service supports this interaction, enabling components to provide notification that a conflict has occurred for which a resolution is sought (see PYRAMID Technical Standard, Ref. [1], section Component Composition).

Brokering: To support the conflict resolution process, the Conflict Resolution component determines the traceability of the demands that are in conflict and applies its defined rules and strategies for brokering a resolution and, if necessary, performing arbitration (see below). Typically, a resolution to the conflict is sought at the lowest level possible in the General Conflict Resolution Pattern with higher levels only becoming involved if necessary, but a variety of strategies might be deployed depending on the nature of the components, demand, conflict or the wider system context. In seeking to broker a resolution to the conflict, the Conflict Resolution component may interact with components at any level of the General Conflict Resolution Pattern. This may involve a request to seek an alternative solution that achieves a refinement goal that, either directly or indirectly, will alleviate the conflict.

This refinement goal is provided by the subordinate level and expressed in terms of the change needed to the demand placed on the subordinate level that would alleviate the conflict, for example Component C might identify a change in the requirement placed on it by Component B that would resolve the conflict (such as the timing of the request for access to a resource). In this way, components at any level of abstraction are engaged in the conflict resolution process in their own terms. Where a component fails to find a suitable alternative solution to meet a refinement goal, it may itself be requested to identify changes to the requirements or constraints placed on it that would allow it to do so. These modifications or relaxations when identified then provide refinement goals for superior levels of the General Conflict Resolution Pattern.

When a conflict is in the process of being brokered, multiple components may be simultaneously engaged to refine their solutions. These refinements may only go part way to resolving the conflict individually, but collectively may be sufficient to resolve it. It is expected that Component C will confirm that a solution has been identified that satisfies the full revised requirement set before:

- Confirming its final solution(s).
- Notifying the Conflict Resolution component that the conflict has been resolved.
- Considering any additional requirements or constraints that might have emerged during the process of conflict resolution.

If Component C still fails to find an acceptable solution, then further iteration via the Conflict Resolution component will be required.

Arbitration: Where alternative solutions cannot be found to resolve a conflict some form of arbitration may be undertaken either facilitated, or directly implemented, by Conflict Resolution. For example, the component that is the source of the conflicting demand may be asked to prioritise one or more of the demands (which can be seen as determining an alternative solution to its own requirement).

In some cases however, there may be no common source of the conflicting demands and therefore no clear arbitration authority. This might occur where components internally generate demands, for example:

- A component's background process may continuously monitor for a specific event, which when observed results in a demand being placed on another component,
- A component may have predefined requirements or constraints that limit its freedom of action.

Such internally generated demands might then conflict with demands resulting from a planned activity with a different origin. The result is the absence of a common source to act as arbitration authority. In the absence of such an arbitration authority, or due to other circumstances, Conflict Resolution may apply its own arbitration rules, for example:

- To apply criteria for immediate arbitration in certain critical situations (e.g. in the case of a high priority demand or time-critical conflict).
- To identify a specific arbitration authority, such as an operator, that should be referred to under various conditions.
- If a deadlock situation occurs.
- To apply criteria to act as a final arbiter where a conflict cannot be resolved by any other means.

A.2.3.5.4.2 Conflict Resolution Component

The Conflict Resolution component is responsible for facilitating a resolution to conflicts between demands and if necessary arbitrating on such conflicts. In order to facilitate a resolution, the Conflict Resolution component must determine the traceability of the demands that are in conflict and understand the rules and strategies for resolution.

The Conflict Resolution component has two key responsibilities:

- **Brokering**: This involves mediating a resolution to demand conflicts, so that available capability can be utilised in the appropriate way. In order to facilitate conflict resolution, the Conflict Resolution component may need to determine the full traceability of the demands that are in conflict, from the demand placed on the recipient component of the requirement (i.e. the component tasked with handling the requirement in conflict) to the overarching requirement from which that demand was derived. There are various mechanisms by which the Conflict Resolution component could determine such traceability, but the PRA does not define any particular approach. The Conflict Resolution component must also understand the policies, rules and strategies for resolving conflicts. The schemas associated with these rules and strategies are a core aspect of the subject matter of the Conflict Resolution component. Such schemas may use information about the nature of the conflict, the components involved in the conflict, the current context, and the priorities of requests (which could be obtained from the components involved in conflict), to define the approach to undertaking resolution (for example the sequence of component interactions). With an understanding of the brokering rules and strategies, and the traceability of demands, Conflict Resolution is able to interact with other components, system users, or other systems to facilitate conflict resolution. This may involve triggering one of the following:

    - Re-planning to determine an alternative solution to meet a defined goal (this could include, for example, the addition of an activity to replenish resources, such as refuelling).
    - Identify relaxations to the requirements or constraints placed on the component that would be necessary to enable that component to determine a solution to meet a defined goal.

  The nature of component interactions with Conflict Resolution is such that each component can understand the conflict in its own terms, with refinement goals exchanged directly between adjacent layers in the General Conflict Resolution Pattern.

- **Arbitration**: This involves making prioritisation decisions where conflicts cannot be resolved by other means, such as re-planning. Prioritisation decisions may be made at a common point of origin of the requirements or constraints from which the conflict stemmed (this will be identified by Conflict Resolution through determining the traceability of the conflict as described in the broking process above). The rules defined in Conflict Resolution may also define an external authority from whom arbitration should be sought, such as an operator. In certain circumstances however, it may be that the Conflict Resolution component is required to arbitrate, for example where no common point of origin is identified. This may include critical situations, for example related to system safety, where clear prioritisation rules are defined, or in deadlock or timeout situations. Here again, the Conflict Resolution component has defined policies, rules and strategies for arbitration.

Any recipient of a demand may place onward dependencies. Such dependencies mean that the recipient of a demand may be the originator of other demands. The refinements necessary to resolve a conflict may thus span multiple components and outside actors, all of which may need to interact with Conflict Resolution to resolve the conflict fully. The management of multiple types of conflict may be handled in several ways by an Exploiting Programme. Component extensions or data driving may be used to define bespoke rules for resolving a particular conflict (e.g. involving a specific recipient). Multiple instances of Conflict Resolution may be used to manage different parts of the system, for example a particular instance could be dedicated solely to managing a single resource type. The number of independent instances of the Conflict Resolution component and which components each instance will manage will be decided by the Exploiting Programme. Similarly, the PRA does not define the nature of interactions between instances, which are considered internal to the PRA component definition and out of scope of the PRA.

A.2.3.6 Feasibility Assessment

Components develop and execute solutions to the requirements provided to them within their given constraints. A component will monitor its own solution (during both planning and execution) to determine whether it remains feasible to achieve its requirements.

After having planned the solution, available resources may change, equipment failures may occur, or there may be changes in the external environment. Such changes could alter the feasibility of a solution within the agreed constraints. Components are allowed (even expected) to adapt their solutions as long as they remain within the parameters defined by the requirements.

If at any point it is discovered that a solution is no longer feasible and no alternative is available, this must be reported to the originator of the requirement. This enables the non-achievable requirements to be re-considered.

At the point a solution is no longer feasible, a new solution space may need to be generated to allow a new solution to be planned. The new solution space will link requirements, dependencies, constraints and resource usage. However, there may come a point when the system has to declare the requirement cannot be met.

A.2.3.6.1 Pattern Overview

The Determine Feasibility use case (see Appendix B: Use Cases) shows how a component will determine the feasibility of carrying out the solution.

Four services are used in feasibility reporting:

- The Requirement service accepts incoming requirements and can report feasibility back.
- The Solution_Dependency service allows the placing of requirements and captures the reported feasibility of them being fulfilled.
- The Constraint_Dependency service enables components to manage constraint dependencies, and can report if the constraint will be breached if it is enforced.
- The Authorisation_Dependency service allows components to request an authorisation, and can report if it is feasible.

Additionally, the following services are needed to support the feasibility assessment:

- The Constraint service accepts incoming constraints.
- The Information_Dependency service acquires contextual information.
- The Capability_Evidence service captures the current and predicted capability of other areas of the system.

A.2.3.6.2 Use of the Pattern

Figure 33: Feasibility Assessment Service Dependencies shows the evidence that components may monitor to determine whether it is feasible for the solution to achieve the requirement it was devised to meet. If new evidence is received, it will result in the feasibility of the solution being reassessed and a new solution sought if the original is no longer feasible or the solution can no longer deliver a quality that meets the required measurement criteria. To determine the new solution, a component may assess all contributing factors before identifying a preferred solution (see Figure 139: Change in the Achievability of Dependent Component Trigger, Figure 137: New Requirement Trigger, and Figure 140: Change in the Constraints Trigger).

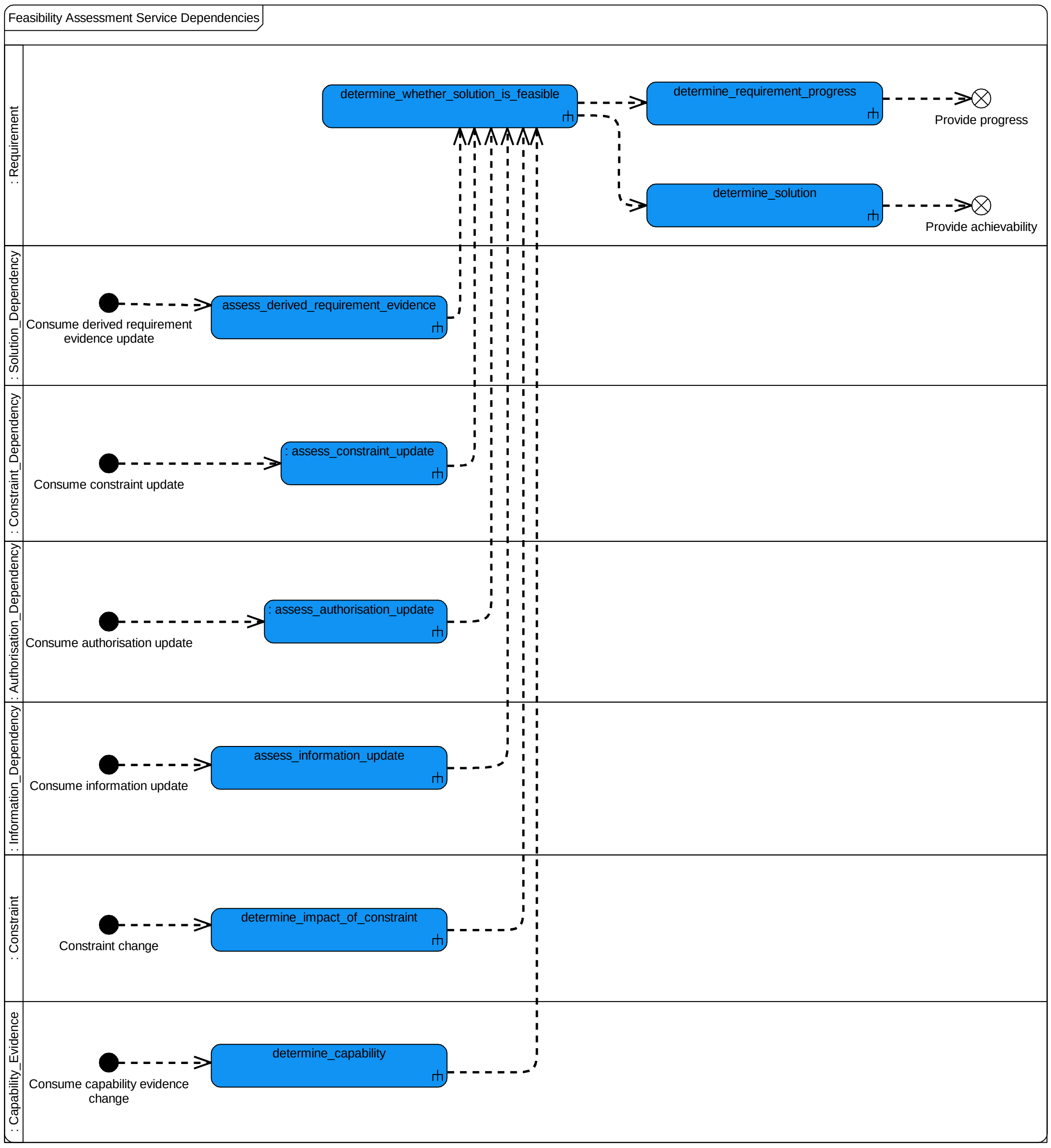

Feasibility Assessment Service Dependencies
: Requirement
determine_whether_solution_is_feasible
determine_requirement_progress
Provide progress
determine_solution
Provide achievability
: Solution_Dependency
Consume derived requirement evidence update
assess_derived_requirement_evidence
: Constraint_Dependency
Consume constraint update
: assess_constraint_update
: Authorisation_Dependency
Consume authorisation update
: assess_authorisation_update
: Information_Dependency
Consume information update
assess_information_update
: Constraint
Constraint change
determine_impact_of_constraint
: Capability_Evidence
Consume capability evidence change
determine_capability

Figure 33: Feasibility Assessment Service Dependencies

A.2.3.7 Achievement and Progress Reporting

It is desirable to assess the satisfaction of requirements in greater detail than a simple ‘complete’ or ‘not complete’ state. By monitoring progress and achievement of each solution, the system can provide more information on the completion of the requirements.

A.2.3.7.1 Pattern Overview

The Fulfil Requirement use case (see Appendix B: Use Cases) addresses achievement and progress reporting, particularly the Figure 141: Fulfil Requirement scenario. Achievement and progress are reported against a requirement, so the Requirement service is used, supported by Solution_Dependency service where derived requirements are involved.

The Generic Interfaces section, within the PYRAMID Technical Standard, Ref. [1], expresses a generic definition of progress and allows achievement reporting to be defined consistently between components.

A.2.3.7.2 Use of the Pattern

Reporting is against a requirement and falls into two categories:

- Achievement: An achievement report indicates whether the requirement has been achieved or not. The report also includes what has been achieved, is currently being achieved or is expected to be achieved and may include a description of the solution’s quality. This report does not have to directly report on the required deliverable and may include concepts broader than quality. See the Generic Interfaces section, within the PYRAMID Technical Standard, Ref. [1], for more information.
- Progress: an indication of progress against the requirement measured against a given criterion. A progress report may identify what has been achieved against the requirement, e.g. 50% complete.

The following example shows how achievement reporting and progress reporting can be achieved.

Figure 34: Achievement and Progress Reporting Structure shows a simple example where the Assigned Component has identified derived requirements which have been passed on to two Contributing Components which, in turn, depend on two more components.

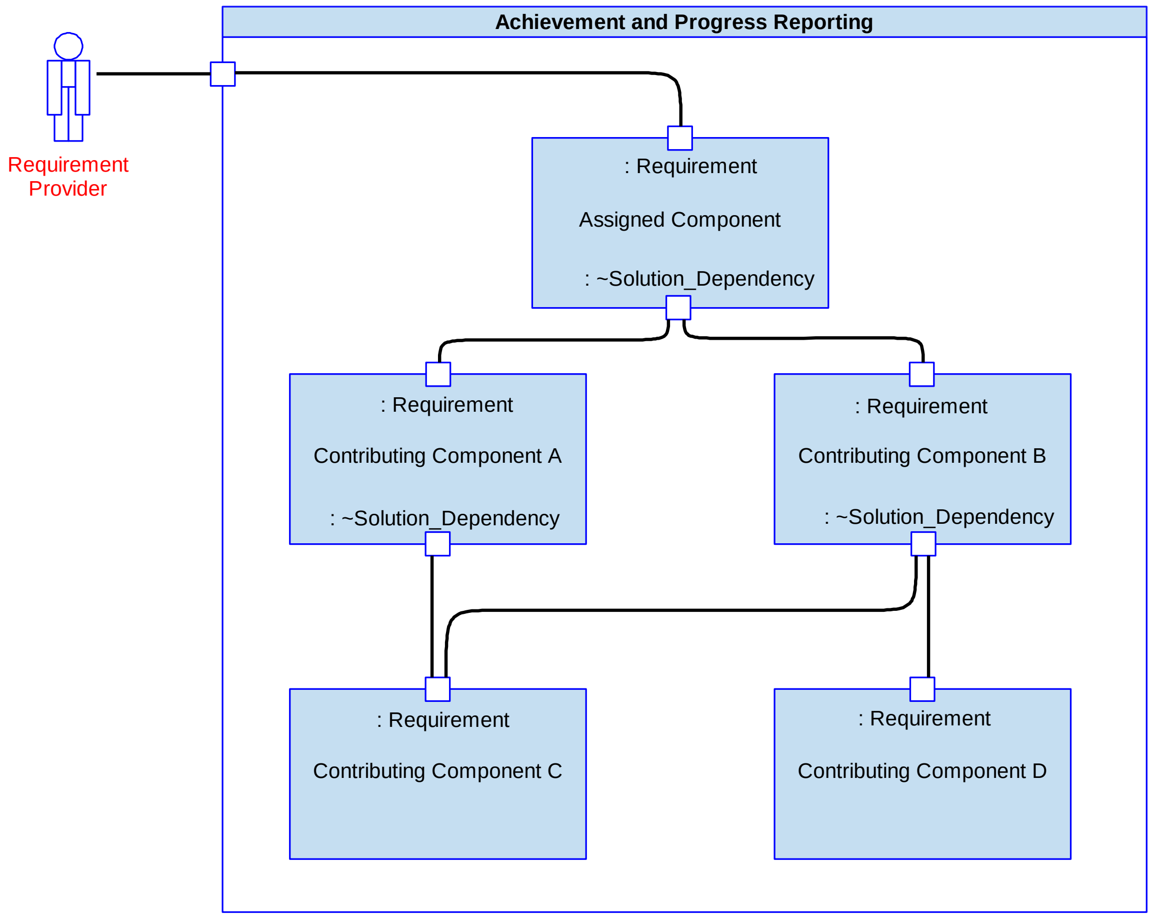


Figure 34: Achievement and Progress Reporting Structure

Figure 35: Progress Reporting Behaviour shows the progress reporting behaviour associated with the example. The Requirement Provider asks for a progress report on a requirement that the Assigned Component has been instructed to fulfil.

Each component, when asked about its progress, first checks progress against requirements it has passed on to components it depends on. This chain is complete when the request reaches a component which can fulfil all of the requirements it is allocated without further assistance and is able to report its own progress.

Progress reports are then passed back up the dependency chain, each component interpreting the reports from its dependencies in terms of its own requirements.

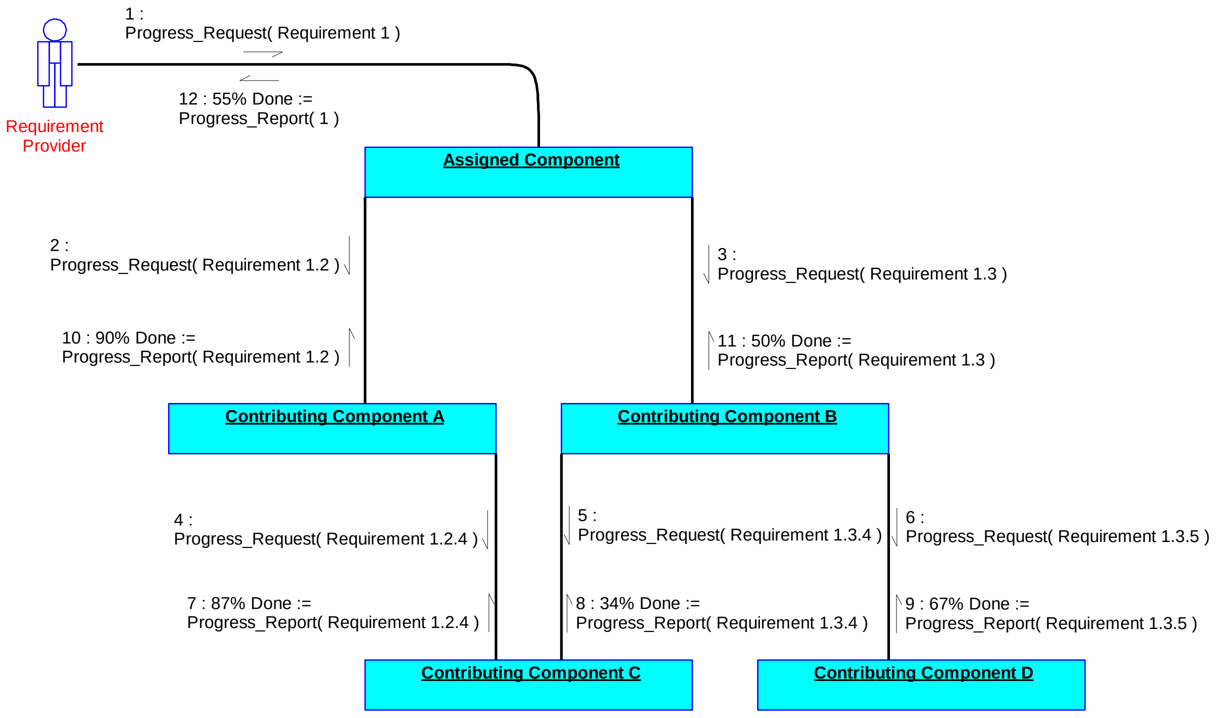
The details of the solution (how the components achieve their requirements) are not part of the progress report.

Reporting of achievement of the requirement can be achieved in a similar manner to progress reporting, where rather than reporting progress towards achieving the requirement, each component reports whether or not it has met its requirement. This can also include statements about whether a component is still on track to meet its requirement (i.e. if the solution to meet the requirement remains feasible).

Achievement of the requirement reporting may also be supplemented with more specific reporting of what has been achieved, is currently being achieved, or is expected to be achieved. For example, irrespective of whether it meets the required threshold demanded by the requirement, the current quality of the required deliverable may be reported.

The component that set the requirements is then able to use these reports to determine if the requirements that it needs to satisfy have been achieved or to determine any necessary supplementary information, such as the specific quality that has been achieved.

Progress reporting could also be implemented as a component outputting progress on a cyclic or event basis, rather than only upon a request.



1 : Progress_Request(Requirement 1)
12 : 55% Done := Progress_Report(1)
Requirement Provider
Assigned Component
2 : Progress_Request(Requirement 1.2)
3 : Progress_Request(Requirement 1.3)
10 : 90% Done := Progress_Report(Requirement 1.2)
11 : 50% Done := Progress_Report(Requirement 1.3)
Contributing Component A
Contributing Component B
4 : Progress_Request(Requirement 1.2.4)
5 : Progress_Request(Requirement 1.3.4)
6 : Progress_Request(Requirement 1.3.5)
7 : 87% Done := Progress_Report(Requirement 1.2.4)
8 : 34% Done := Progress_Report(Requirement 1.3.4)
9 : 67% Done := Progress_Report(Requirement 1.3.5)
Contributing Component C
Contributing Component D

Figure 35: Progress Reporting Behaviour

A.2.3.8 Performance Analysis

A component can specify a required quality as part of a solution dependency. Additionally, components can report their predicted and achieved qualities of any measurement criteria relating to the achievement of a requirement (such as a track position accuracy) as well as the progress against achieving a requirement and the current feasibility of the solution. This reporting allows analysis and assessment of performance against a requirement.

Since reporting occurs through a series of components with solution dependencies on one another, analysis can be carried out at different levels of requirement abstraction. Performance can be assessed by each component in an operational system. This approach also captures changes in performance due to changes in the external environment, such as an enemy platform employing a new tactic after its previous tactic was defeated.

Performance analysis during operation allows for underperforming solutions to be dynamically improved, or (if compatible with other requirements) for satisfactory solutions to be dynamically optimised. This may include generating qualitatively different solution dependency requirements as part of re-planning rather than refining existing ones.

The results of the performance analysis may also be recorded to aid non real time analysis (see the Operational Support PYRAMID concept).

In some circumstances the reason for the system underperforming may be due to an anomaly, such as a health issue or cyber attack, resulting in a reduction in capability which will need to be taken into account when re-planning a solution (see the Capability Management PYRAMID concept).

A.2.3.9 Solution Replanning

When a solution that has been selected for execution, or one that is in progress, has become unfeasible, a solution replan becomes necessary. The following sections present two example scenarios requiring solution replanning, one for a solution that is replanned following a change in the feasibility of the solution actions, the other is for a solution that suffers a loss of resource.

A.2.3.9.1 Action Replan

Figure 36: Solution Action Replan shows a simple example where the currently selected solution to a requirement is Solution 1. Solution 1 is made up of three actions, of which Action 3 is using Action 3.1 of the two available sub-options. When Action 3.1 becomes unavailable, it requires the solution to be replanned for Action 3 to use Action 3.2.

Action 3 can be updated for the replanned solution, without needing to involve the Requirement Provider, providing Solution 1 still meets the requirement and has not changed the cost or quality.

If Solution 1 no longer meets the requirement, then the alternative solution (i.e. Solution 2) will be considered.

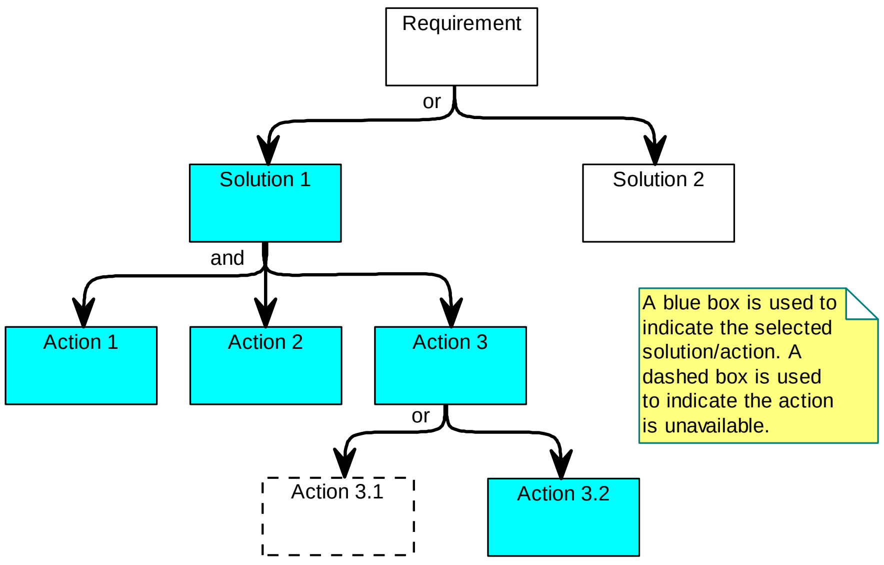

Requirement
or
Solution 1
Solution 2
and
Action 1
Action 2
Action 3
or
Action 3.1
Action 3.2
A blue box is used to indicate the selected solution/action. A dashed box is used to indicate the action is unavailable.

Figure 36: Solution Action Replan

A.2.3.9.2 Resource Replan

Figure 37: Solution Resource Plan shows an example of a requirement that cannot be solved by a single component. Consequently, the solution depends on two Components, X and Y. Both Components have selected required resources; Components X and Y have selected Resources A

and C respectively. Both Components, X and Y, have alternative solutions that can use other resources, i.e. Resources B and A respectively.

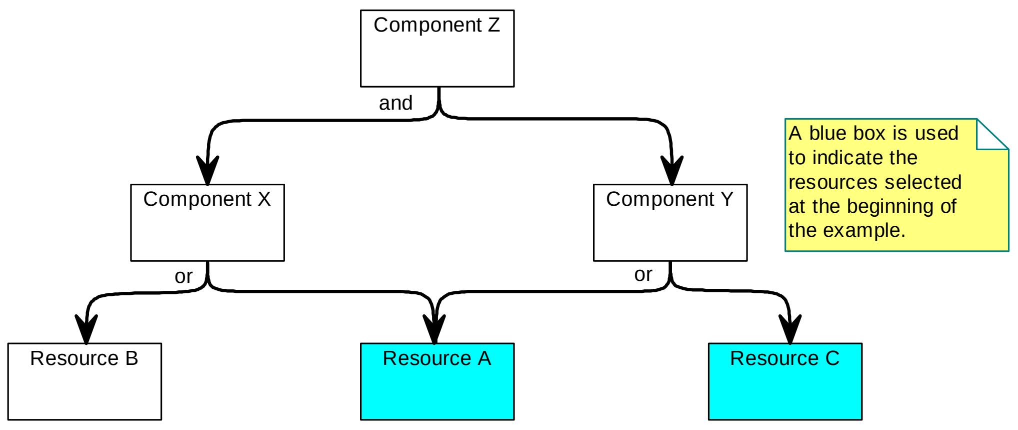

Component Z
and
Component X
Component Y
or
or
Resource B
Resource A
Resource C
A blue box is used to indicate the resources selected at the beginning of the example.

Figure 37: Solution Resource Plan

In this example, Resource C becomes unavailable, thus Component Y must use its alternate solution which depends on Resource A, currently used by Component X, to fulfil its requirement. The desire of Component Y to use Resource A is not directly expressed to Component X, but the allocation is linked through the solution dependencies. The Resource Management PYRAMID concept shows how conflicts can be identified allowing a replan of Component X's activity so that it doesn't require the use of Resource A.

Figure 38: Solution Resource Replan shows the new resource plan with Component X no longer using Resource A. Component Y is now allowed to use Resource A and the new solutions from both components X and Y are accepted by Component Z and the required resources allocated accordingly.

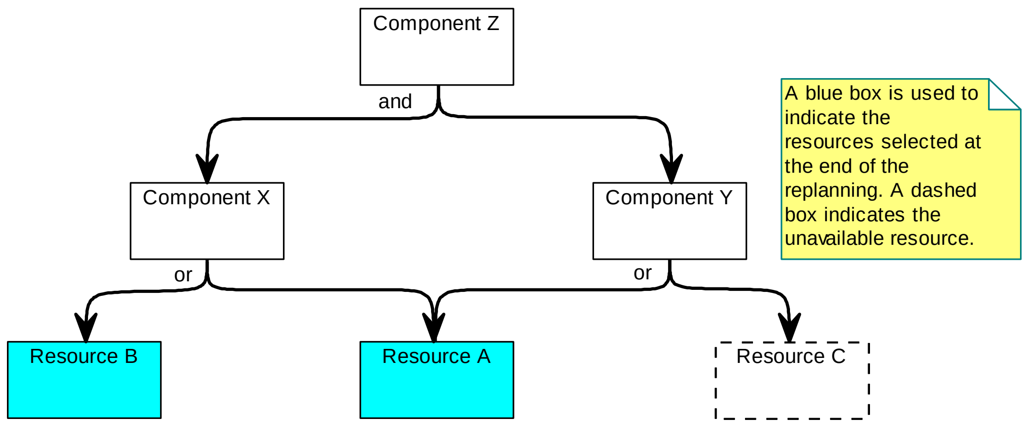

Component Z
and
Component X
Component Y
or
or
Resource B
Resource A
Resource C
A blue box is used to indicate the resources selected at the end of the replanning. A dashed box indicates the unavailable resource.

Figure 38: Solution Resource Replan

A.2.3.10 Examples of Rigid and Dynamic Dependencies

A.2.3.10.1 Setting Up a Sensing Task

The system is given the mission objective of searching an area for a particular class of objects. The objective is broken down into two tasks: make a transit flight to the area, and perform a search. The search involves flying over the area, using available sensors to scan the area, and processing sensor data to identify objects. This use case covers the setting up of the two linked tasks.

A.2.3.10.1.1 Pre-Conditions

The search objective has been assigned to an aircraft.

A.2.3.10.1.2 Flow

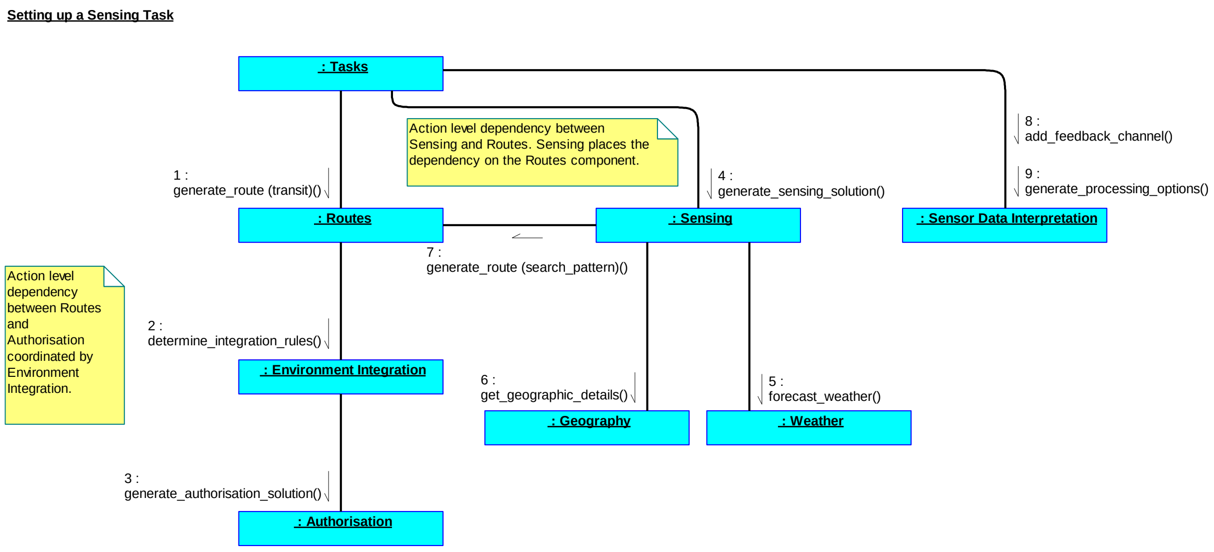

Setting up a Sensing Task
: Tasks
: Routes
: Sensing
: Sensor Data Interpretation
: Environment Integration
: Geography
: Weather
: Authorisation
Action level dependency between Sensing and Routes. Sensing places the dependency on the Routes component.
Action level dependency between Routes and Authorisation coordinated by Environment Integration.
1 : generate_route (transit)()
2 : determine_integration_rules()
3 : generate_authorisation_solution()
4 : generate_sensing_solution()
5 : forecast_weather()
6 : get_geographic_details()
7 : generate_route (search_pattern)()
8 : add_feedback_channel()
9 : generate_processing_options()

Figure 39: Setting up a Sensing Task

The Tasks component plans the transit task, using Routes to find a transit solution. A route is proposed which would require authorisation to fly through a restricted zone (determined by Environment Integration). This is identified as a dependency for the transit, and Authorisation determines that it will be possible to request clearance.

The Tasks component plans the search task, using Sensing to find a sensing solution. Based on the details of the search task, together with knowledge of the search area (from Geography) and the weather forecast (from Weather), a sensing solution is proposed. This involves flying a particular search pattern, which is placed as a dependency on Routes.

The Tasks component also uses Sensor Data Interpretation to plan the tactical processing required to analyse the outputs from the sensing actions. A dynamic dependency is created between Sensor Data Interpretation and Sensing to manage the search as data is received.

A.2.3.10.1.3 Post-Conditions

The tasks are ready for execution.

A.2.3.10.2 Carrying Out a Sensing Task

The system is given the mission objective of searching an area for a particular class of objects. This is broken down into two tasks: make a transit flight to the area and perform a search. The search involves flying over the area, using available sensors to scan the area, and processing sensor data to identify objects. This Use Case covers the execution of the two tasks.

A.2.3.10.2.1 Pre-Conditions

The tasks have been set up and the mission is ready for execution.

A.2.3.10.2.2 Flow

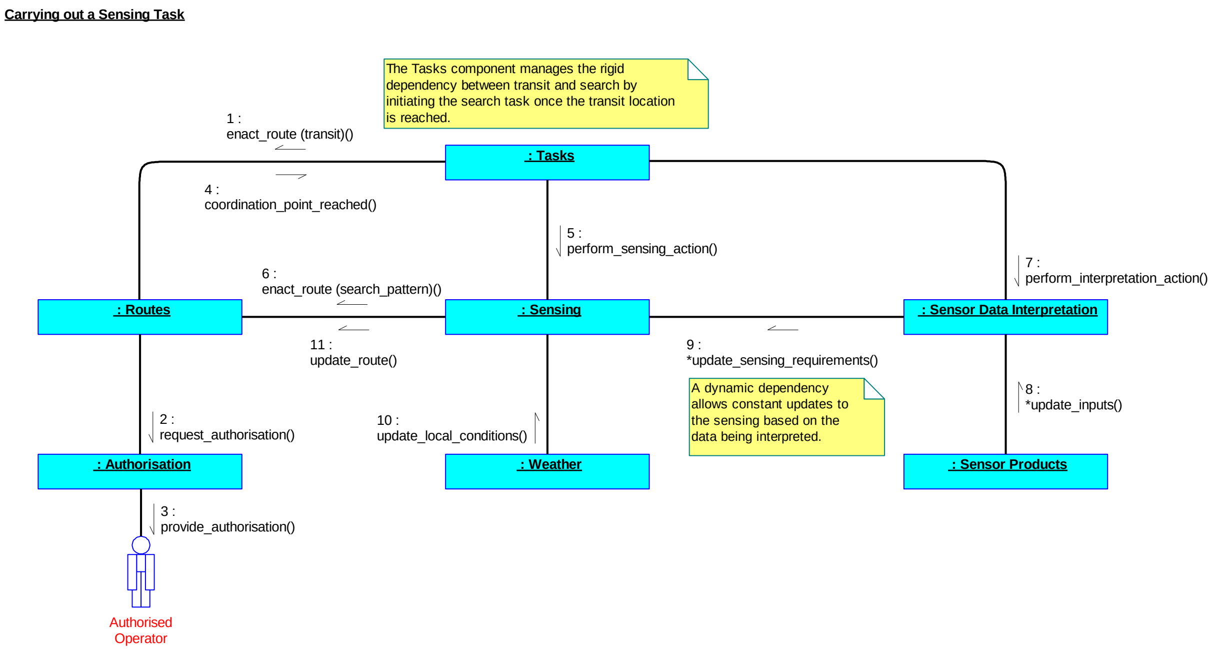

Carrying out a Sensing Task
The Tasks component manages the rigid dependency between transit and search by initiating the search task once the transit location is reached.
1 : enact_route (transit)()
4 : coordination_point_reached()
5 : perform_sensing_action()
6 : enact_route (search_pattern)()
7 : perform_interpretation_action()
: Tasks
: Routes
: Sensing
: Sensor Data Interpretation
11 : update_route()
9 : *update_sensing_requirements()
A dynamic dependency allows constant updates to the sensing based on the data being interpreted.
8 : *update_inputs()
2 : request_authorisation()
10 : update_local_conditions()
: Authorisation
: Weather
: Sensor Products
3 : provide_authorisation()
Authorised Operator

Figure 40: Carrying out a Sensing Task

Tasks manages the dependency between transit and search by initiating the search task once the transit location is reached. Tasks requests that Routes execute the planned transit route. Routes can begin execution of the route as the aircraft is not in the restricted area, but relies on Authorisation obtaining permission to transit through the restricted zone. Authorisation obtains this approval from an Authorised Operator. This allows the Routes component to continue through the rest of the planned route.

Tasks receives an update when the search area has been reached. At this point the search task is initiated. This involves flying the planned search pattern (via a request to Routes), using the sensors as planned, and analysing the sensor data to identify the objects of interest.

As the sensor data from Sensor Products is analysed, Sensor Data Interpretation adjusts the requirements on Sensing (without exceeding the original requirements and constraints) to make sure that the sensors are used in a way that will support the data analysis. This is an example of a dynamic dependency between components as defined in What is Dependency Management?.

Changes reported in Weather conditions are reported to Sensing to determine if this will impact the capability to carry out the search. As the search progresses and sensor data is processed, the parameters fed to Routes can be refined to update the search path.

A.2.3.10.2.3 Post-Conditions

The tasks are complete.

A.2.3.11 Dependency Management Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	PYRAMID Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept identifies the following generic component behaviours in relation to dependency management: • Components capture requirements that fall within their subject matter. • Components determine solutions in order to satisfy the demands placed upon them. • Components identify dependencies when establishing a solution(s) to the requirement placed upon them and generate derived requirements (to be placed on other components). • Components identify where conflicts exist that potentially result in one or more requirements being unable to be fulfilled. • Components support a process of brokering and arbitration, to resolve conflicts, by: • Performing solution refinement to achieve a specified goal (aimed at resolving a conflict). • Determining refinement goals for the demands placed on them (aimed at resolving a conflict). • Making arbitration decisions where their dependent requirements directly conflict. • Components monitor the progress of their own solution and determine whether it remains feasible to achieve the requirement that the solution was intended to satisfy. • Components report their achievement and progress against their assigned requirements. • Components monitor and analyse their performance, enabling dynamic solution optimisation and to support non real time analysis.	The PRA defines the following generic component responsibilities and services in relation to dependency management: **Responsibilities:** • capture_requirements • capture_measurement_criteria • determine_solution • determine_solution_dependencies • coordinate_dependencies • identify_conflict • determine_if_solution_remains_feasible • identify_whether_requirement_is_achievable • identify_progress • determine_quality_of_deliverables **Services:** • Requirement • Dependency_Refinement • Information • Solution_Dependency • Constraint_Dependency • Authorisation_Dependency • Information_Dependency • Broker_Conflict For more information on dependencies relating to constraints, capability and autonomy, please see the Constraint Management, Capability Management and Autonomy PYRAMID concepts respectively.
Specific PRA Component support for the PYRAMID Concept	The PRA separates out the following areas of functionality with regards to dependency management: • Understanding, and application of, the processes and rules for the brokering and arbitration of conflicts.	The PRA defines the following component whose role relates to dependency management: • Conflict Resolution
Exploiter considerations	The following dependency management related considerations are a matter for individual Exploiting Programmes: • The particular approach or mechanism by which traceability of solution dependent information is managed (especially when multiple solutions are available) to i) enable components to distinguish between related sets of requirements, constraints and information (which must be treated collectively) and unrelated sets that can be treated independently, ii) facilitate the resolution of conflicts through brokering and arbitration. • The specific rules and methodologies used for the brokering and arbitration of conflicts.	The PRA does not mandate a specific approach in respect of these considerations.

Table 3: Dependency Management PYRAMID Concept Summary

A.2.4 Autonomy

A.2.4.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Control Architecture
- Constraint Management
- Human-Machine Interface

A.2.4.2 Introduction and Scope

This PYRAMID concept explains what is meant by autonomy in the context of the PRA. It introduces the Authorisation component and describes how this component, acting together with other components discussed in the Authorisation section, can be used to develop and configure a system capable of operating at varying levels of autonomy.

A.2.4.2.1 What is Autonomy?

The PRA regards an autonomous system as one that can develop and implement problem solutions in accordance with pre-defined rules without direct crew control, following the receipt of any necessary authorisation.

A.2.4.2.2 Why is Autonomy Important?

The PRA supports the design and configuration of a system incorporating autonomous behaviour. This gives PRA Exploiters the ability to design for, and control, the levels of autonomy required by an Exploiting Platform.

A.2.4.2.3 Levels of Autonomy

A system may be able to operate at varying levels of autonomy, ranging from fully autonomous to fully manual. The level of autonomy may be constant for a particular Exploiting Platform or mission, or it may change depending on, amongst other things:

- The operating environment.
- The specific task or action being carried out.
- The state of the infrastructure at the time. For example, if communication is reduced or lost the vehicle may be permitted to operate with greater autonomy.
- The state of the vehicle at the time. For example, an automatic response to internal events such as faults may be permissible.
- The workload of the authorised operators. For example, enabling more autonomy during periods of high operator workload.

An action or decision which is autonomous can be considered to have received authorisation. However, there are situations where, although an action or decision is autonomous, there may be a benefit in the operator being informed, for example to maintain their situation awareness and enable them to make decisions or take action when required. Other actions or decisions may require greater operator involvement, for example to choose between a number of potential solutions generated by the system or to act as arbitration authority for a solution (see the Dependency Management PYRAMID concept for details about arbitration), and to enable this the system would need to provide the operator with necessary information. Benefits of this operator involvement may include:

- A greater ability to respond to unforeseen circumstances and a rapidly changing environment.
- The ability to apply judgement and interpretation for non-deterministic rules (e.g. rules of engagement) or scenarios.
- Potentially reduced development costs of some aspects of the system (where automation technology is prohibitively expensive).
- Improved engagement/situation awareness for the operator.
- A safe alternative during early stages of development to allow automation technology to be matured.
- Mitigation for some failure conditions.

The level of autonomy is therefore linked to authorisation and interaction with the operator. There are a number of models and frameworks available that provide methods of quantifying and measuring the level of autonomy at which a system can operate, and it is down to the Exploiting Programme to determine which is appropriate. An example of how levels may be applied to a search task is described in the Styles of Interaction section.

A.2.4.2.4 Introduction to Authorisation

Every activity carried out by the system requires some form of operator authorisation. This may be built into the system by design (in the form of general rules), or as an explicit authorisation for a task or specific action. The authorised operator's act of placing a requirement (and accepting the cost of the solution) may constitute authorisation for the task as a whole. However, even a task being carried out with general authorisation, or under a high level of autonomy, may need separate authorisation for a specific action, such as jettisoning fuel, depending on the rules that are in force. For such an action, the system must ensure it has authorisation before carrying out the action.

A.2.4.3 Overview

The PYRAMID concept illustrates how the components enable a PYRAMID deployment to be configurable for different levels of autonomy either at design time or during operation. The PYRAMID concept illustrates how the different styles of interaction can be achieved, and how authorisation can be associated with either whole tasks, or specific actions that form part of a task solution.

A.2.4.4 Component Composition

This PYRAMID concept identifies a number of generic component behaviours in relation to Autonomy. These behaviours are formally captured as component responsibilities in the PYRAMID Technical

Standard, section Component Composition, along with associated component services. These responsibilities and services are also summarised in Table 4: Autonomy PYRAMID Concept Summary.

Appendix B: Use Cases demonstrates how these responsibilities and services support component interaction in relation to Autonomy, specifically with regards to authorisation.

A.2.4.5 Context

Context is information that is likely to change the interpretation of observed facts or behaviour in some way. For example, the presence of a tank in a particular area may or may not be interpreted as a threat depending on the understood posture of the tank's owner.

Context is not just an issue for operators; the more complex and autonomous a system is, the greater importance context has for the system as well. Systems which develop solutions (either for automated implementation or proposing to an operator) need to understand context so that the solution is appropriate for the current context.

Within the PRA, each component is responsible for a particular subject matter, and associated contextual information belongs to that component and will be provided when necessary to support decision making throughout the system. For any given component, its context is established by any information it can obtain. This can be from other components, users, or other systems. This is illustrated in Figure 41: Establishing Context which shows Component X consuming contextual information to enable decision making within its defined autonomy limits (or alternatively determine the need for authorisation). Human-machine interoperability is key to autonomy as it must be ensured that the human and machine have access to the same data, and that their interpretations of that data are made in the same context. This is illustrated in Figure 42: Sharing Context.

Figure 41: Establishing Context and Figure 42: Sharing Context illustrate how components can support both sides of a system and operator interaction. The independence of components means that the same services which support Component X can also support the involved HMI components.

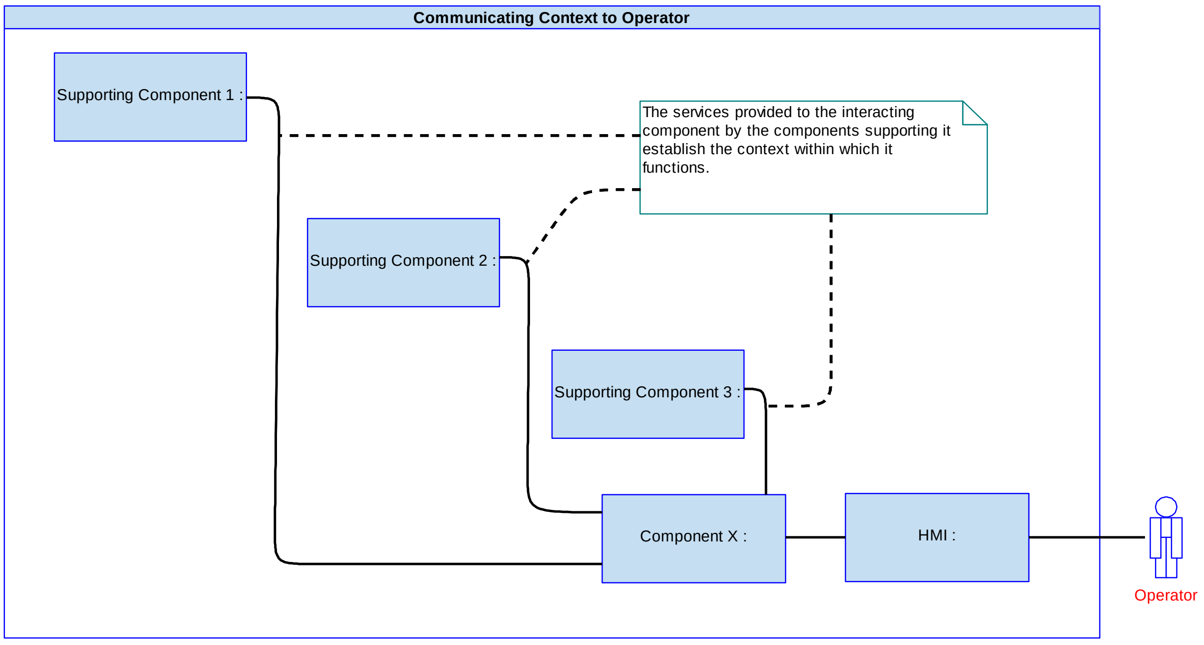

Communicating Context to Operator
Supporting Component 1 :
Supporting Component 2 :
Supporting Component 3 :
Component X :
HMI :
Operator
The services provided to the interacting component by the components supporting it establish the context within which it functions.

Figure 41: Establishing Context

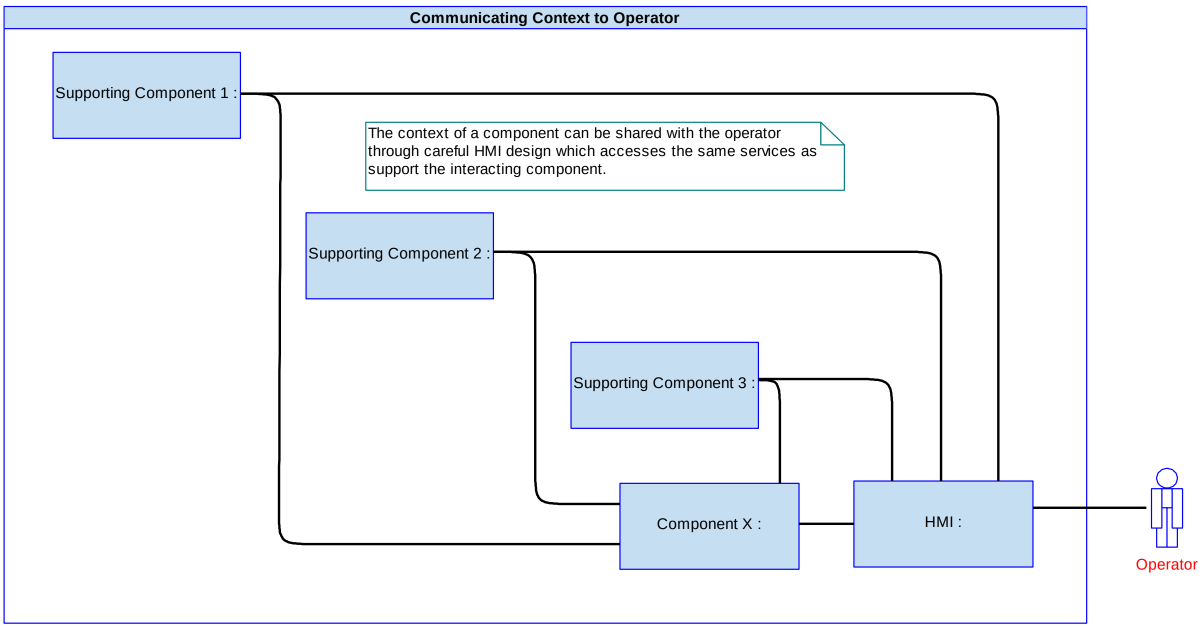

Communicating Context to Operator
Supporting Component 1 :
Supporting Component 2 :
Supporting Component 3 :
Component X :
HMI :
Operator
The context of a component can be shared with the operator through careful HMI design which accesses the same services as support the interacting component.

Figure 42: Sharing Context

A.2.4.6 HMI and User Roles

The system Human-Machine Interface (HMI) will need to support the requirement for authorisation of any actions, and any associated interactions with users. For example, a fully autonomous system, without a requirement to provide feedback or present options to an authorised operator, or obtain authorisation for a specific action (such as jettison fuel), will not require any HMI to support autonomy. The system may of course require HMI for other reasons.

However, if there is a requirement for operator involvement, for example to obtain authorisation, inform an authorised operator of an action, or provide solution options, the HMI must support this interaction, enabling the appropriate information to be presented to the authorised operator and, where necessary, translate their response into instructions to be enacted by the system.

A.2.4.7 Authorisation

The need for authorisation of an activity is satisfied by a number of components acting together:

- The Authorisation component is responsible for obtaining authorisation when it is explicitly required. What constitutes authorisation, from the perspective of the Authorisation component, will depend on the Exploiting Platform, but the authorisation must be made in a meaningful and informed way, in line with the Human-Machine Interface PYRAMID concept.
- The Operational Rules and Limits component captures general rules which can constrain a component's operational ability, which may result in the component having to obtain authorisation to continue.
- Components are responsible for understanding their autonomy remit and identifying when authorisation is required before they can carry out or request specific activities. A component's specific activities may:
    - Be inherent to a components design and be naturally allowed as part of its normal behaviour.
    - Require authorisation to be obtained before performing the activity, e.g. this could involve seeking authorisation, checking that pre-authorisation is in place or checking that previous authorisations are still valid.
    - Be assumed possible until the component is informed that they are prohibited. This restriction would be typically indicated by the application of a constraint, possibly provided via the Operational Rules and Limits component or from another component.

  Where a component has identified that authorisation is required, this authorisation would be achieved by the component liaising with the Authorisation component. The Authorisation component's responsibilities lie with capturing requirements for authorisation from another source and developing an authorisation solution.

Authorisation for a specific activity may be granted through virtue of authorisation having been granted to a broader activity that it contributes to. In this case either authorisation may be implied for a component when it receives a requirement to be implemented, or confirmation of authorisation may be required to be obtained from the Authorisation component - where in this case the Authorisation component would already have the relevant authorisation available. The latter option may, for example, be desirable to increase system safety integrity. However, in some cases, whilst authorisation for a broader activity may have been granted, dedicated authorisation for specific activities may still be needed. For example, authorisation may have been granted to carry out activities contributing to attacking a target; however, separate authorisation may still be required prior to committing to weapon release against the target. In this case the Authorisation component would need to obtain this separate authorisation.

It is likely that the requirement for authorisation will vary depending on conditions. A potential solution to a requirement placed on a component may capture this variability as a measure of the quality of the proposed solution. For example, if the solution to a search task runs a risk of requiring authorisation to enter a restricted zone, this can be captured against the solution as a low robustness to change.

When granting authorisation for a mission plan, or a particular activity within a plan, an authoriser may wish to place conditions upon the authorisation such that it is only valid if the specified conditions are met. For example, it would only be valid within a particular time window or only with target data quality above a defined threshold. Within the PRA, conditions associated with an authorisation will be held with that authorisation, however it will be a responsibility of the Authorisation component to compare the current state with the boundaries of the conditions.

A.2.4.8 Styles of Interaction

In the example where the system has the task to make a transit to reach an area to be searched, the interaction with the operator is governed by the way the HMI has been set up, how the operator is choosing to use it and by behavioural rules that are active at the time.

Depending on the context within which the task is being solved there may be different solutions to the task according to the criteria that are considered. The Mission Plan may require the fastest route or the shortest. It may focus on probability of success or robustness to change. If the Mission Plan does not include these criteria, the operator may be left to decide. Measurement of generated solutions against such criteria will allow the operator, or the system, to select the most appropriate solution. Operating at the highest levels of autonomy the system will look for a solution that best matches the criteria, possibly informing the operator and enabling them to reject the solution. At lower levels of autonomy the system may look for individual solutions for each criterion to allow the operator to select between them. These interactions between the operator and the system will usually be coordinated by the high-level components Objectives and/or Tasks but can take place with any layer of the control architecture, as shown in Figure 18: Authorised Operator Interactions.

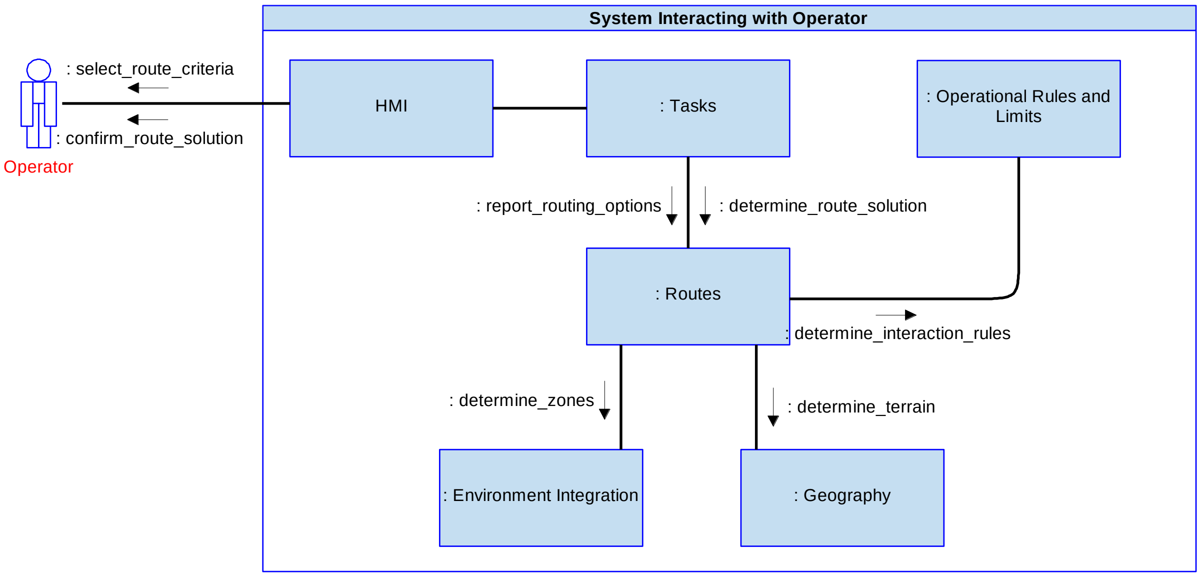


Figure 43: Example of System Interacting with Operator

Figure 43: Example of System Interacting with Operator shows how the operator and the system may cooperate to determine a route to transit to a search area.

The system will behave differently depending on the requirements for authorisation and interaction with an authorised operator, and how the HMI is set up. Some examples of how the system and operator may interact are given below:

- **Full autonomy:** The system generates the best solution it can find and enacts it, providing feedback to the operator as required.
- **Action unless revoked:** The system generates the best solution it can find and presents it to the operator. Unless the operator objects, the system enacts the planned task.
- **Advice and action if authorised:** The system generates solutions, presents them to the operator and highlights the recommended one. The operator approves a solution and the system enacts the planned task.
- **Advice:** The system generates solution options by using different criteria. The operator selects a solution (after refining the criteria, if required) and the system enacts the planned task.
- **Advice only if requested:** The system generates solution options to meet criteria given by the operator. The operator selects a solution (after refining the criteria, if required) and the system enacts the planned task.
- **None:** The system plays no part in generating or selecting solutions.

A.2.4.9 Autonomy Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Autonomy Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PRA supports the design and configuration of systems incorporating autonomous behaviours. All PRA components are responsible for understanding their autonomy remit and the conditions under which authorisation for its actions is required. The level of a component’s autonomous behaviours may be data driven or determined through interactions with other components. For example, authorisation limits could be imposed as constraints or determined based on contextual information acquired from other components, users, or other systems.	The PRA defines the following generic component responsibilities and services: **Responsibilities:** • capture_constraints • capture_autonomy_remit • determine_authorisation_dependencies **Services:** • Authorisation_Dependency • Constraint • Information_Dependency For more information regarding how PRA components manage and generate constraints and dependencies, see the Constraint Management and Dependency Management PYRAMID concepts.
Specific PRA Component support for the PYRAMID Concept	The PRA separates out the following areas of functionality associated with autonomy: • Knowledge of authorisation policy and authorisation entities and the process of obtaining authorisation. • Understanding of operational rules and the system limits that derive from these rules (including constraints on autonomous behaviour). • The management and curation of information required for interactions between systems and their users (which enable user authorisation to be obtained). • How to represent the conveyance and perception of information between the system and an HMI user.	The PRA defines the following components whose role directly relates to autonomy: • Authorisation • Operational Rules and Limits • HMI Dialogue • Information Presentation For more information on Operational Rules and Limits, HMI Dialogue and Information Presentation see the Constraint Management and Human-Machine Interface PYRAMID concepts.
Exploiter considerations	The following autonomy related considerations are a matter for individual Exploiting Programmes: • The choice of framework within which system autonomous behaviours are defined, e.g. autonomy levels. • The specific policies or rules that define and constrain the autonomous behaviours of any individual component or the system as a whole.	The PRA does not mandate a specific approach in respect of these considerations.

Table 4: Autonomy PYRAMID Concept Summary

A.2.5 Health Management

A.2.5.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Capability Management
- Control Architecture
- Cyber Defence

A.2.5.2 Introduction and Scope

This PYRAMID concept explains the mechanisms provided by the PRA for the management of system health. The PYRAMID concept introduces the Anomaly Detection and Health Assessment components and describes their interaction with capability assessment at all levels of the architecture.

A.2.5.2.1 What is Health Management?

In the context of the PRA, a system's health describes its ability to perform its intended function within specified limits.

Health can be described for a complete system or for any of its constituent parts. If any of the hardware elements that make up the system have failed or been damaged, the system's health is reduced because it will be unable to perform, or to perform fully, the functions that rely on that hardware element. Looked at more broadly, the overall system may still have full capability because, by design, built-in redundancy means it can achieve the same function using alternative resources. Nevertheless, at the local level, a part of the system has reduced health, and overall, the system has become less robust.

Health management is the process of detecting an anomaly, identifying the element (or elements) that gave rise to it and then providing health information to support capability assessment. The health management process also reacts appropriately to a health problem to keep the system safe and/or allow it to fulfil its mission as well as providing life and usage information, which supports maintenance and system availability.

A.2.5.2.2 Why is Health Management Important?

Incorporating health management into a system may contribute to the safety of the system by reducing the chance that failures will lead to avoidable hazards. For mission-critical functions, good health management may allow the system to achieve its mission goals in the presence of failures, leading to a higher mission success rate. Health management also contributes to the availability and maintainability of a system by providing health data to inform maintenance decisions and activities.

Health management applies at different system levels:

- At the fleet level, by changing working practices, maintenance procedures, manufacturing standards, or other systematic changes.

- At the asset level, by performing or triggering maintenance actions to restore the health of the asset.
- At the mission level, by finding alternative ways to deliver a mission capability using different resources.
- At the resource level, by reconfiguring the infrastructure to use redundant or backup resources. This kind of response may be automatic and performed without extensive analysis of the problem.

A.2.5.3 Overview

The PYRAMID concept shows how the components contribute to health management by:

- Detecting that an anomaly (system not being in an expected state or exhibiting unexpected behaviour as a result of not being in that state) has occurred that may indicate a health problem.
- Identifying the system element that gave rise to the anomaly or anomalies as a consequence of its failure.
- Providing health information to support capability assessment as discussed in the Capability Management PYRAMID concept.
- Reacting appropriately to a health problem to keep the system safe and allow it to fulfil its mission goals if at all possible.
- Providing life and usage information to support maintenance and system availability.

A.2.5.4 Health Assessment

Except for automatic responses to a failure, which may be carried out instantaneously where a very fast response is essential, the first step in good health management is to understand the failure. This is health assessment.

The system is monitored, at every level of detail, to determine whether it is behaving as expected. Any unexpected behaviour needs to be explained. It could be caused by:

- An external condition such as weather. This may require a controlled response to accommodate the condition.
- A software problem. Unknown software artefacts (including but not limited to bugs) may exist within the software, or software may be compromised by cyber attack.
- A hardware failure.
- Recovery from a failure. In some hardware there is the potential for failures to recover (e.g. blockages sometimes clear and leaks may get plugged), software can be reset and some systems may accommodate failures internally.

Health assessment is the process of distinguishing between these cases, i.e. determining whether the unexpected behaviour is related to the state of system hardware. For hardware failures (and recoveries) it identifies:

- The change in health of the system, including how it can be expected to progress. This may then be used to determine the effect on the capability of the system. The Capability Management PYRAMID concept covers this.
- The hardware element (or elements) that has incurred the change.

This information allows appropriate decisions to be made at resource, mission, asset and fleet levels. Where the health of a system element is difficult to assess directly, its life (time since initial operation) or usage (how, or how much it has been used) may be used as a proxy for the health of a system and can inform maintenance requirements.

A.2.5.4.1 Detection of Anomalous Behaviour

Anomalous, or unexpected, behaviour results in the system entering an unexpected state. An unexpected state may be as simple as an error message, or it may be an inconsistency between two values. Such inconsistencies may only be apparent when different parts of the system are compared: each part of the system may appear to be consistent when viewed locally. Anomalies therefore need to be considered at every level of the system, not just at the interface with hardware.

For some hardware elements it may be possible to detect discrepancies that are an early sign of degradation that could get worse. In such cases it will be desirable to monitor for such discrepancies, where the element they relate to is still operating within its specification, but not in the expected range. Local small discrepancies may also be part of a wider pattern that could be a sign of a cyber attack or a physical problem such as overheating.

A.2.5.4.2 Failure Identification and Prognostics

In order to make maintenance decisions, and also to have a more precise knowledge of the effect of a health change on its capability, the system needs to identify the hardware element that has failed or been damaged.

In an integrated system, the failure of a hardware element may cause problems in any part of the system that relies on that element, however distantly. Ideally, each hardware element would be monitored closely and any failure identified immediately and unambiguously. This may indeed be the case in highly instrumented safety-critical systems, where an anomaly is clearly related to a specific functional failure. In other cases, however, it may not be possible to get the specific information that would make a health assessment easy.

Where the assessment of health is unable to provide an answer that is sufficiently certain or specific to inform either maintenance or operational decisions, it may identify additional information that could improve the answer. If this includes requesting a BIT to be run, the test will be managed by the Test component as shown in the Test PYRAMID concept.

In health management, prognostics is the use of health data to predict the progression of a failure or damage over time and with use. In the PRA, prognostics is not treated separately, but is regarded as a forward-looking aspect of health and capability assessment. Prognostics may also be done as part of operational support when the prediction is too long-term, or too uncertain, to inform decisions within a mission. However, many failures progress either too fast or too slowly, or unpredictably (or not at all) to be considered; so in many cases the best assumption will be that the current hardware condition, and hence capabilities that rely on it, will continue unchanged.

A.2.5.4.3 Health Data

Health data includes all data that is required as input for assessing the health and health related capability of a system. It can include, but is not limited to:

- Hardware and software configuration data.
- Fault and error codes (BIT reports).
- Sensor data (including, but not restricted to, specific sensors for monitoring health and structural integrity).
- System control, command and mode data.
- Consumables data.
- Usage data (for life and usage monitoring).
- Manual measurements (requested and volunteered).

The collation of health data that needs to be available for exploitation outside the system is coordinated by the Information Brokerage component in accordance with the Storage and Recording and Logging PYRAMID concepts.

The exploitation of health data for fleet management and maintenance is discussed in the Operational Support PYRAMID concept.

A.2.5.5 Health Components

This section describes a framework for the use of components to support health management. It will help component developers and implementers understand the context in which the health management components are intended to work.

A.2.5.5.1 Health Management of a Single Resource

This section shows a simple example with a single instance of the Health Assessment component assessing the health of a single resource. Larger, more complex systems will generally require many instances or extensions of Health Assessment working together, this is discussed further in Hierarchy of Health Components.

In Figure 44: Health Management of a Single Resource, the Resource Provider component, Effectors, manages a hardware resource, an actuator. The Resource User component, Mechanical Positioning, makes use of that resource to carry out an activity such as opening a door to allow the release of a store. (See Resource Management PYRAMID concept, section Overview for a description of the featured terms Resource User and Resource Provider.)

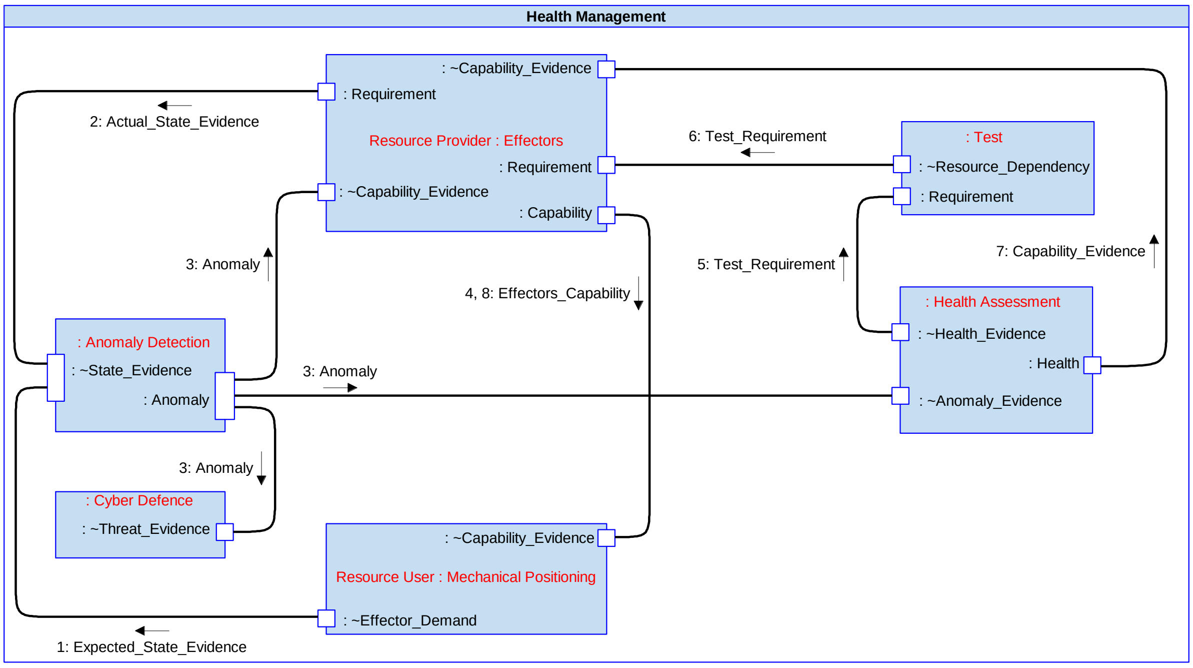


Figure 44: Health Management of a Single Resource

In this example, the Resource User, Mechanical Positioning provides the expected state of the resource, following issue of a command to change the state of the actuator. The actual state of the actuator comes from a separate sensor via the Resource Provider, Effectors.

The Anomaly Detection component uses this information to compare the actual state to the expected state. In this case, it determines that the actual and expected states do not concur and this is classified as an observed anomaly. Anomaly Detection publishes the observed anomaly to Effectors, Cyber Defence and Health Assessment.

Effectors revises the determination of its capability and informs the Resource User.

The Cyber Defence component will consider whether the anomaly could be a sign of a cyber attack, see the Cyber Defence PYRAMID concept for further discussion on detection of cyber attacks.

The Health Assessment component requests the Test component to initiate the relevant built in tests, see the Test PYRAMID concept for further discussion on testing support within the PRA. The BIT results can be used to inform a maintenance assessment so that the affected part can be repaired or replaced without extensive diagnostic activity.

Health Assessment informs Effectors that there may be a change to its dependent capability. The information from Health Assessment about the health issue may allow the effect on capability to be determined more precisely.

Effectors assesses its capability based on the information received and any effects on its capability due to failures are provided to Mechanical Positioning. The capability assessment will also be informed by local anomalies and/or failures of the hardware they control. Components further from the hardware will base their assessment on the change in capability of other components that they rely on, see the Capability Management PYRAMID concept.

Additional considerations:

- Where hardware is concerned, under most circumstances a fault or error message from the hardware constitutes evidence of an anomaly.
- Some resource components may have the ability to reconfigure their resource in response to an identified failure in their hardware so that the resource remains available.
- Many health issues are of concern to authorised operators, and may require warnings or alerts to be raised. Whilst this and the pertinent HMI components are not shown here, the relevant information needing to be provided to an operator should be sourced from the component responsible for the information. Therefore, health information will be supplied by Health Assessment and capability information will be supplied by the component responsible for the relevant capability. These pieces of information may be used in unison, by the HMI, to provide the operator with an understanding of the capability implications of a health issue.

A.2.5.5.2 Hierarchy of Health Components

Although it is possible to architect a deployment of the PRA such that there is a health assessment solution involving a single instance of the Health Assessment component with knowledge of the entire system, this solution means that every change to any part of the system will require that component to be updated. This is usually only practical for small, simple systems. For larger systems a distributed solution using multiple instances or extensions of the Health Assessment component is more flexible, with a hierarchy of Health Assessment components working together in a coordinated way to identify where in the system the problem actually lies. There should be no assumption that the real cause of the problem is in the component that was detected as being in an anomalous state. A distributed solution will require the Health Assessment components in the system to have a consistent approach to carrying out fault diagnosis (reasoning about cause and effect), for example by having a generic implementation of the Health Assessment component. Individual instances and extension components can be tailored by data driving, if safety concerns allow, with information about the hardware element(s) they are concerned with. Health Assessment components at higher levels will have a more abstract view of the system and will collate and interpret the reasoning of lower level Health Assessment components. The dependencies between hardware elements, which determine how failures will propagate, inform the way Health Assessment components are connected to each other.

Health is relevant throughout the system, and problems in hardware resources may be observed at any level of the system. The hierarchy of Health Assessment components maps to the Exploiting Platform’s system decomposition (e.g. the physical equipment, physical structural elements, assemblies, etc.) but with due consideration of the functional and structural separation of equipment. For example, a piece of equipment may perform a primary function, such as computing, but may also have a role in providing structural strength. A fault in the equipment may only impact on one of these roles and so both roles may need to be considered independently. This may fall within the responsibility of separate instances or extensions of the Health Assessment component.

A hierarchy of Health Assessment components enables higher level Health Assessment components to:

- Diagnose faults that can only be diagnosed when considering the system broadly, such as where a combination of faults in different parts of a system is a key indicator of a common cause.

- Collate an overall view of the health of a system where:
  - Mapping specific hardware health issues to a system level health issue is necessary, such as identifying that a faulty physical structural element has health implications on if, or how, the overall airframe can be used.
  - System health is more complex than the sum of its healthy parts, such as where designed redundancy enables a system to remain operational despite faults occurring.

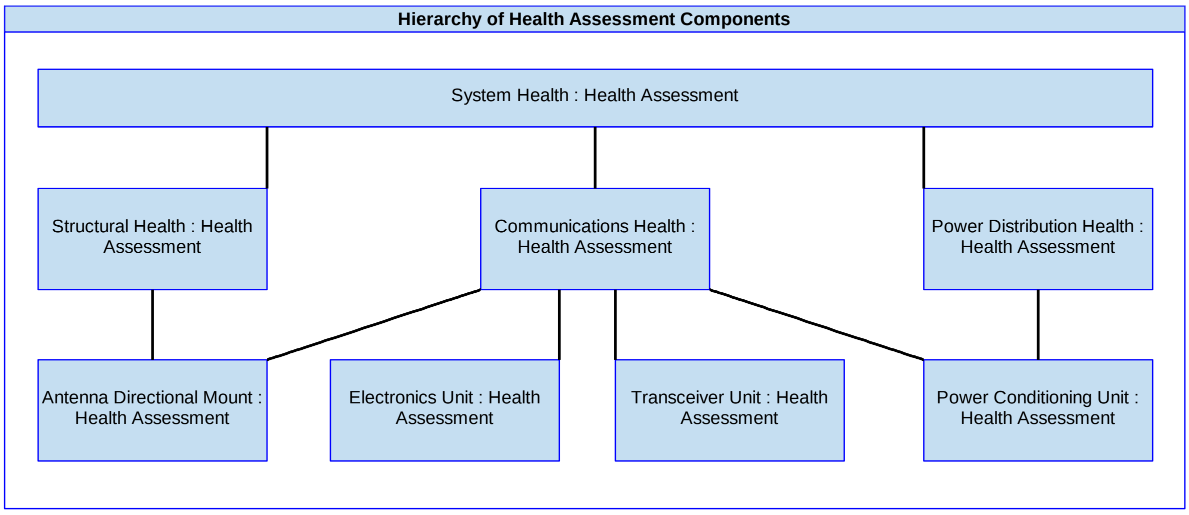

Hierarchy of Health Assessment Components
System Health : Health Assessment
Structural Health : Health Assessment
Communications Health : Health Assessment
Power Distribution Health : Health Assessment
Antenna Directional Mount : Health Assessment
Electronics Unit : Health Assessment
Transceiver Unit : Health Assessment
Power Conditioning Unit : Health Assessment

Figure 45: Hierarchy of Health Assessment Components

Figure 45: Hierarchy of Health Assessment Components shows the Health Assessment components associated with a communications array. There is a Tx/Rx unit for communications, a supporting electronics LRU, a dedicated power conditioning unit, and a directional mount that is controlled by the pointing computer. Each part of the system has a Health Assessment component monitoring performance and reporting system health to the relevant interested parties. The Health Assessment component associated with each of these elements interprets the collated health information and reports further, allowing a single overseeing point of understanding for the whole system health, which can use this information, along with anomalies reported by the Anomaly Detection component, to diagnose faults holistically. Anomaly Detection (not shown here) is relevant throughout the system, since anomalies may be detected at any level. In this example, an anomaly (loss of communications capability) is traced through the power distribution and communications related Health Assessment components to a failure in the power conditioning unit.

Anomaly Detection reasons about anomalies in behaviour that can be influenced by mechanical, electrical or software system elements, unexpected changes in structural elements of the system, and the environment (including hostile actors). The hierarchy of Anomaly Detection components will therefore map to behaviours, some of which may relate to PYRAMID components and how they interact.

Health Assessment components are intended to communicate with each other up, down and across the health hierarchy when necessary, in order to identify the root cause of an anomaly, if that anomaly is the result of a health change. However, the hierarchical relationship between each of the Health Assessment components is largely dependent on the relationships between physical items of equipment. It is expected that the health assessment made by one instance of the Health Assessment component can contribute to the health assessment in another instance of the Health Assessment component. An example of this could be where two different pieces of equipment are physically co-located, such as radio and GPS/INS LRUs. An anomaly indicates that the GPS/INS health has started to deteriorate but it may be a direct result of the radio overheating (root cause). In this instance aspects of both items may be handled by individual instances of Health Assessment components and may need to share health information in order to correctly diagnose the root cause.

The detection of anomalies and assessment of health within the computing infrastructure is performed by the computing infrastructure, such as by the operating system, and so is not performed by the PRA components. However, the results from these processes within the computing infrastructure may be used by the PRA components to aid the reporting of issues, such as to system operators, or to aid reasoning about the wider contributions, implication, or causes of such issues within the system as a whole.

A.2.5.5.3 Cooperation of Components

Figure 46: Cooperation of Health Management Components shows an example of how health management components work together to determine the cause of an anomaly which is not apparent in small parts of the system viewed locally.

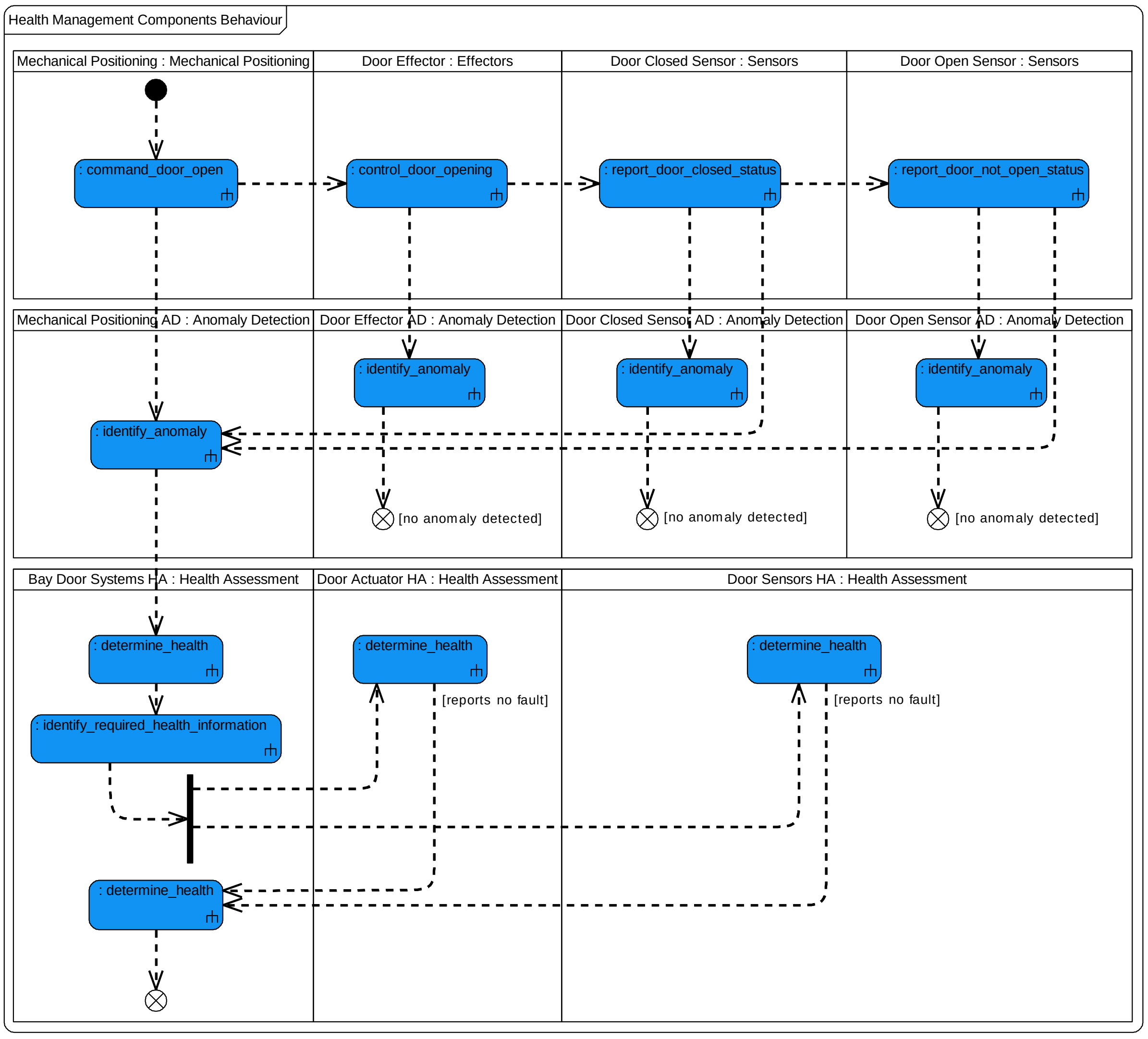

Health Management Components Behaviour
Mechanical Positioning : Mechanical Positioning
Door Effector : Effectors
Door Closed Sensor : Sensors
Door Open Sensor : Sensors
: command_door_open
: control_door_opening
: report_door_closed_status
: report_door_not_open_status
Mechanical Positioning AD : Anomaly Detection
Door Effector AD : Anomaly Detection
Door Closed Sensor AD : Anomaly Detection
Door Open Sensor AD : Anomaly Detection
: identify_anomaly
[no anomaly detected]
Bay Door Systems HA : Health Assessment
Door Actuator HA : Health Assessment
Door Sensors HA : Health Assessment
: determine_health
: identify_required_health_information
[reports no fault]

Figure 46: Cooperation of Health Management Components

A jettison solution has been identified which requires a door to be opened so a store can be jettisoned. The Mechanical Positioning component is responsible for controlling the door opening by commanding the Effectors component interfacing with the door. No anomalies are detected by Anomaly Detection in the Door Effector component, nor the Anomaly Detection for the Sensors components; when viewed locally they appear to be working as expected. The Mechanical Positioning Anomaly Detection component receives the actual state of the door from the Door Closed and Door Open Sensors components. The Door Closed Sensor reports that the door is closed, and this is consistent with the report that the door is not open from the Door Open Sensor component. However, the Mechanical Positioning Anomaly Detection component is also aware that the door was commanded open.

This anomaly in the state of the door is passed to the higher level Bay Door Systems Health Assessment component for assessment (note that the hierarchy of Health Assessment components does not necessarily align with the hierarchy of Anomaly Detection components). In an attempt to determine if the cause of the anomaly is health related, the Bay Door Systems Health Assessment requests the Health Assessment components associated with the doors to provide information on the health status of the door effector and door sensors equipment. The Health Assessment components lower in the hierarchy determine that the necessary hydraulics and power have been supplied to the effector, and that no fault is found in either the effector or sensors equipment. The Bay Door Systems Health Assessment assesses the evidence and determines that it is likely that either both door sensors have failed or that the door effector has failed. Using its subject matter knowledge of door effector and door sensor failures (including likelihood), Bay Door Systems Health Assessment determines that the most likely cause of the anomaly is a failure of the door effector. Although not shown on this diagram, this would result in the Door Actuator Health Assessment component being informed that the door effector is not working, and the health reported to interested parties for capability assessment and maintenance purposes.

A.2.5.6 Understanding of Capabilities

Capability assessment, as explained in the Capability Management PYRAMID concept, is the responsibility of each component. When an anomaly is detected, associated components are notified. The anomaly may be the result of a failure somewhere else in the system, but it still represents an unexpected state as far as the operation of the component is concerned. The component may have already observed the unexpected behaviour as part of its usual monitoring of its actions. Either way, the anomaly may be directly related to the capability of the component, especially where there is a high degree of monitoring in the hardware, as well as its own behaviour, as is often the case in safety-critical areas.

This is shown in Figure 44: Health Management of a Single Resource, where an anomaly in the state of a resource has been identified and notification of the anomaly is provided to the resource provider, which revises the determination of its capability as a result. This information is then available in the hierarchy of capability dependencies and components that rely on that capability can understand how their own capability has been changed as a consequence.

Figure 44: Health Management of a Single Resource also shows that a detected anomaly is simultaneously reported to the local Health Assessment component, which will identify the hardware element that has caused the problem. That information will feed back into the associated component. The information from Health Assessment about the health issue may allow the effect on capability to be determined more precisely. If the failed hardware is associated with a different component to that which was detected as being in an anomalous state, the update from the Health Assessment component could be its first indication of the problem. Simultaneously, the Cyber Defence component assesses the anomaly to see if it is a sign of a cyber attack.

Capability assessment is the responsibility of each component whereas Health Assessment and Anomaly Detection are individual components in their own right. When viewing Figure 44: Health Management of a Single Resource it is important to view it in this context. An unambiguous view of how the capability assessment functionality is being implemented can be viewed in Capability Management PYRAMID concept, Figure 48: Capability Example: Structure.

A.2.5.7 Insulating the System from Hardware Dependencies

One of the goals of the health management components is to insulate the other parts of the system from having to understand the details of their dependency on hardware, especially as it relates to failure and damage.

Exploiting Programmes will want the freedom to replace equipment with minimal impact on the system. Unless the replacement is identical, it may have different failure modes, even if it is designed to provide the same functional behaviour and interface. In that case, the changes to accommodate various failure modes can be restricted to the Anomaly Detection and Health Assessment components, allowing other components that deal with the equipment to be unchanged.

Similarly, only the health management components need to understand the dependency of component software on the computing infrastructure. This allows the software to be redistributed without affecting the bulk of the components.

This separation makes exploitations of the PRA more portable and resilient to obsolescence.

A.2.5.8 Implementation

For a PYRAMID deployment, health components will need tailoring, but they must also work together to diagnose failures. A desirable approach is to have a generic implementation of Anomaly Detection and Health Assessment which can be data-driven to accommodate the specific details of the implementation, including types of hardware and how they are connected. Bespoke implementations may also be required in some specialised areas. In all cases, the connectivity of the implementation will determine which components Anomaly Detection and Health Assessment report to.

ISO 13374 Ref. [13] covers condition monitoring and diagnostics of machines. It is a widely-recognised industry standard for fault diagnosis, and the PRA must support implementations that conform to it. Components have been designed to correspond to the six processing blocks identified in the standard, as shown in Table 5: Correspondence between ISO 13374 and the PRA. This correspondence enables a deployment of the PRA to conform to the data processing and communication protocols required by the standard.

ISO 13374	PRA
Data Acquisition	Not in the PRA: this will be handled by infrastructure.
Data Manipulation	This may be handled by infrastructure or by Anomaly Detection as required by the deployment.
State Detection	Anomaly Detection component.
Health Assessment	Health Assessment component.
Prognostic Assessment	Health Assessment component.
Advisory Generation	Out of scope. This processing block is concerned with the maintenance and use of equipment, which would be part of fleet management.

Table 5: Correspondence between ISO 13374 and the PRA

A.2.5.9 Health Management Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Health Management Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept does not identify any generic component behaviours related directly to health management. However, PRA components support the process of health management as follows: • By providing configuration information* (e.g. for an equipment to which the component interfaces) that may be used in detecting an anomaly, or assessing current/future health including usage and ageing. • By providing status information* (e.g. the loss of a function within an equipment to which the component interfaces) that may be used in detecting an anomaly or assessing current/future health. • By providing usage information* (e.g. for an equipment or subsystem to which the component interfaces or in respect of wider aspects of airframe usage), that may be used in predicting health trajectories and to identify maintenance requirements. • By providing contextual information* (e.g. the environmental conditions) that may be used in detecting an anomaly, or assessing current/future health including usage and ageing. • By reassessing and reporting their capability in response to notification of an anomaly, or health problem, associated with an equipment to which the component interfaces. • By predicting their capability progression in response to notification of the expected health trajectory of an external equipment. • By assessing their capability in response to capability evidence and responding to capability shortfalls. * The identified categories of information that PRA components may provide in support of anomaly detection and health assessment is not exhaustive.	The PRA does not define any generic component responsibilities or services specifically for the purpose of health management. However, all components may have provided or consumed services that enable the sharing of subject matter data; this may include the reporting of configuration, status, usage and contextual information, for example. The responsibilities of individual components in respect of assessing and reporting capability is described in the Capability Management PYRAMID concept.
Specific PRA Component support for the PYRAMID Concept	The PRA separates the following areas of functionality that relate to health management: • The detection of anomalies that might indicate a system health problem. • The assessment of the current and projected health of system hardware items. • The coordination of testing in the determination of system health.	The PRA defines the following components, whose role relates to health management: • Anomaly Detection • Health Assessment • Test For more information about the Test component, see the Test PYRAMID concept.
Exploiter considerations	The following health management related considerations are a matter for individual Exploiting Programmes: • The assessment of the current and projected health of computing infrastructure. • The specific approach to system partitioning to enable health management, for example, the use of component variants and instances, and the use of extensions.	The PRA does not mandate a specific approach in respect of these considerations.

Table 6: Health Management PYRAMID Concept Summary

A.2.6 Capability Management

A.2.6.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Health Management
- Cyber Defence
- Dependency Management

A.2.6.2 Introduction and Scope

This PYRAMID concept explains how the concepts of capability assessment and shortfall resolution have been applied to the PRA. It describes what is meant by capability in the context of the PRA and how to achieve consistent capability assessment in a PYRAMID deployment. It also outlines how once identified, the cause of a capability shortfall can be determined, if a shortfall resolution is required, and how shortfall resolution can be achieved at both component and system levels.

A.2.6.2.1 What is Capability?

Capability is the ability to do something, or the ability to perform a particular function. A system's capabilities are derived from the resources it has at its disposal, and the uses it is able to put them to. Higher-level capabilities such as searching depend on lower-level capabilities such as sensing and data processing, which in turn rely on resources like sensors and power. The concept of capability therefore applies at every level of a system, but its expression changes from one level to another.

A system's capability varies with time and internal circumstances, such examples include:

- The designed capability of a resource, such as a sensor, may be restricted by the way it is installed.
- A failure can remove or restrict the capability of a hardware element, which may have consequential effects on higher-level capabilities that depend on the resource.
- A constraint imposed on a component may mean its capability is restricted, again with possible effects on higher-level capabilities.

Whilst capability is not based on external factors, external factors can affect the capability of equipment, such as where a cold environment causes icing of an actuator, thereby, degrading or removing its capability. The capability is always the inherent capability of the system or an element of the system; however, any strategy for recovering the capability may need to recognise the underlying reason for the loss of capability, which may be an external factor.

This is in contrast to cases where external factors limit the system performance or what it can achieve in specific scenarios, without inherently changing the system capability; such as in the case where an adversary jams a sensor, preventing accurate target detection, despite the sensing equipment remaining fully functional.

Within the PRA, capability is distinguished from achievability of a requirement and feasibility of a solution. Achievability and feasibility are described in the Dependency Management PYRAMID concept. The achievability of a requirement is influenced by the feasibility of its planned solution, which relies on the available capability. Capability assessment does not take resource allocation into account; theoretical capability is determined, ignoring how a resource may already be allocated.

A.2.6.2.2 Why is Capability Management Important?

Every part of a PYRAMID deployment is responsible for making decisions at its own level. To do that correctly, it needs to know not only the requirements and constraints that are imposed on it, but also the capability that it has at its disposal. Components can be deployed in many configurations, so capability must be assessed and managed in a way that reflects the configuration of an Exploiting Platform. This applies both to the way it is designed and to how it is affected by run-time events such as reconfigurations and failures. A dynamic assessment of capability throughout the system means that every part of the system has an up-to-date view of its capabilities that will enable it to make decisions. Additionally, having the ability to address shortfalls in capability will allow the system the greatest chance of ensuring that the highest possible level of system capability is available when required.

A.2.6.3 Overview

The principle behind this PYRAMID concept is that every component in the PRA is responsible for assessing its own capability. This applies to components at every layer in the control architecture (see Control Architecture PYRAMID concept). Each component bases its assessment of its capability on its own inherent capability (which may be influenced by the data loaded into the component, or the lack of such data) and the capabilities that it relies on. The capabilities that a component relies on includes capabilities provided by other components and other capability dependencies, such as the elements of the execution platform which enable the component to operate and equipment controlled by the component.

Reconfigurations which change the dependencies between components may cause the system's capability to be re-assessed. A failure in hardware may lead to a loss of capability reported by a component. A cyber attack may cause a compromised component to suffer a loss in the capability it can provide. These losses of capability will be reflected as a corresponding loss of capability in the components that rely on these services. Shortfalls or losses in capability may be addressed either at a component level (for example, triggering a piece of hardware to restart with an aim to regain its capabilities) or at a system level which will use the combined ability of multiple components to address an identified shortfall.

A.2.6.4 Component Composition

This PYRAMID concept identifies a number of generic component behaviours in relation to Capability Management. These behaviours are formally captured as component responsibilities in the PYRAMID Technical Standard, section Component Composition, along with associated component services. These responsibilities and services are also summarised in Table 7: Capability Management PYRAMID Concept Summary.

Appendix B: Use Cases demonstrates how these responsibilities and services support component interaction in relation to Capability Management.

Note that these services are also used within the component interaction patterns and examples provided within this concept.

A.2.6.5 Capability Assessment

There is no separate "Capability Assessment" component: each component is responsible for assessing its own capability. Components derive their capability from the resources available to them, either directly or via other components. A resource component's capability is likely to be very heavily tied to the health of the equipment itself, but may relate to other factors. This PYRAMID concept expresses a standard approach which allows components to communicate their capability in a uniform way.

When a component is informed of a change of capability of something it relies on, there are three things it must do:

- If possible accommodate the change so that it can continue to provide its capability to other components as best it can.
- Assess how the change affects the feasibility of any current or planned activities, then report back the activities that are no longer feasible.
- Assess how that change affects its own capability, and make that assessment available to other components.

Only the last of these is covered by this PYRAMID concept. The Dependency Management PYRAMID concept covers the first two.

The reporting of capability assessment is in general not binary; that is, the capability is not just reported as available or not available (although it could be). Instead, different levels of capability (perhaps due to a degradation in capability) can be reported against each aspect of capability that a component reports. This enables a component receiving the results of a capability assessment (i.e. a statement of another component’s capability) to determine its own capability in an accurate way, with the level of dependant capability being taken into account at any particular moment in time.

The Health Management PYRAMID concept explains how the capability of hardware resources is assessed. This is an important source of information for capability assessment.

Figure 47: Capability Interactions shows the standard services that are available on a component that assesses its capability (most components are in this category) and how two such components communicate their capability. Each component has two services:

- A Capability service, which makes its own capability assessment available;
- A Capability_Evidence service, by which it receives information about the capabilities it relies on.

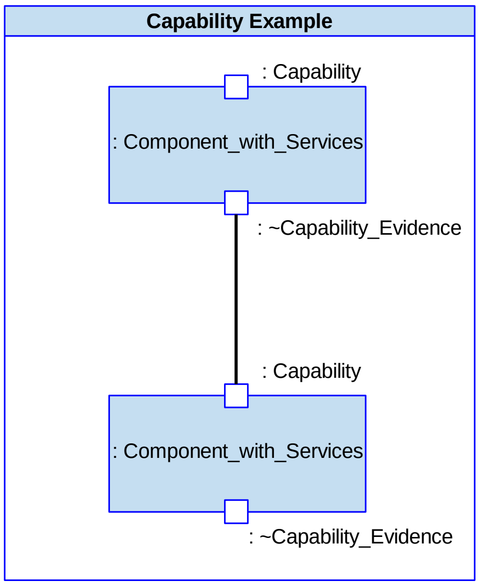


Figure 47: Capability Interactions

Some components (e.g. resource components) will receive information about their resource directly, rather than through the Capability_Evidence service.

Each component assesses its capability in its own terms by understanding how it relies on the components and resources that support it. This pattern is applied repeatedly so that each component can understand, in its own terms, how its capability is affected by changes in the capability of resources.

The Generic Interfaces section within the Technical Standard contains a generic definition of capability that can be specialised to specific components.

The Capability_Evidence service is also used for capability information that comes from the Anomaly Detection, Health Assessment and Cyber Defence components. See the Capability Assessment Example section and there is more information in the Health Management and Cyber Defence PYRAMID concepts.

The Capability service is used to communicate capability information to wherever it is needed: this could include HMI components, for example, as well as components that rely on that capability.

A.2.6.5.1 Capability Assessment Example

This section gives a typical example of how components might work together to provide capability assessment. The two diagrams (Figure 48: Capability Example: Structure and Figure 49: Capability Example: Behaviour) show structural and behavioural features of the example and should be considered together.

A.2.6.5.1.1 Structure

In Figure 48: Capability Example: Structure Tasks has access to an Action Layer component, Action Component W, which can use two resource components, Resource Component X and Resource Component Y, to carry out actions in support of a task. All these components exchange capability information using their Capability and Capability_Evidence services. This is a simple example: in a more realistic case, other action components may also rely on the capabilities provided by Resource Component X and Resource Component Y.

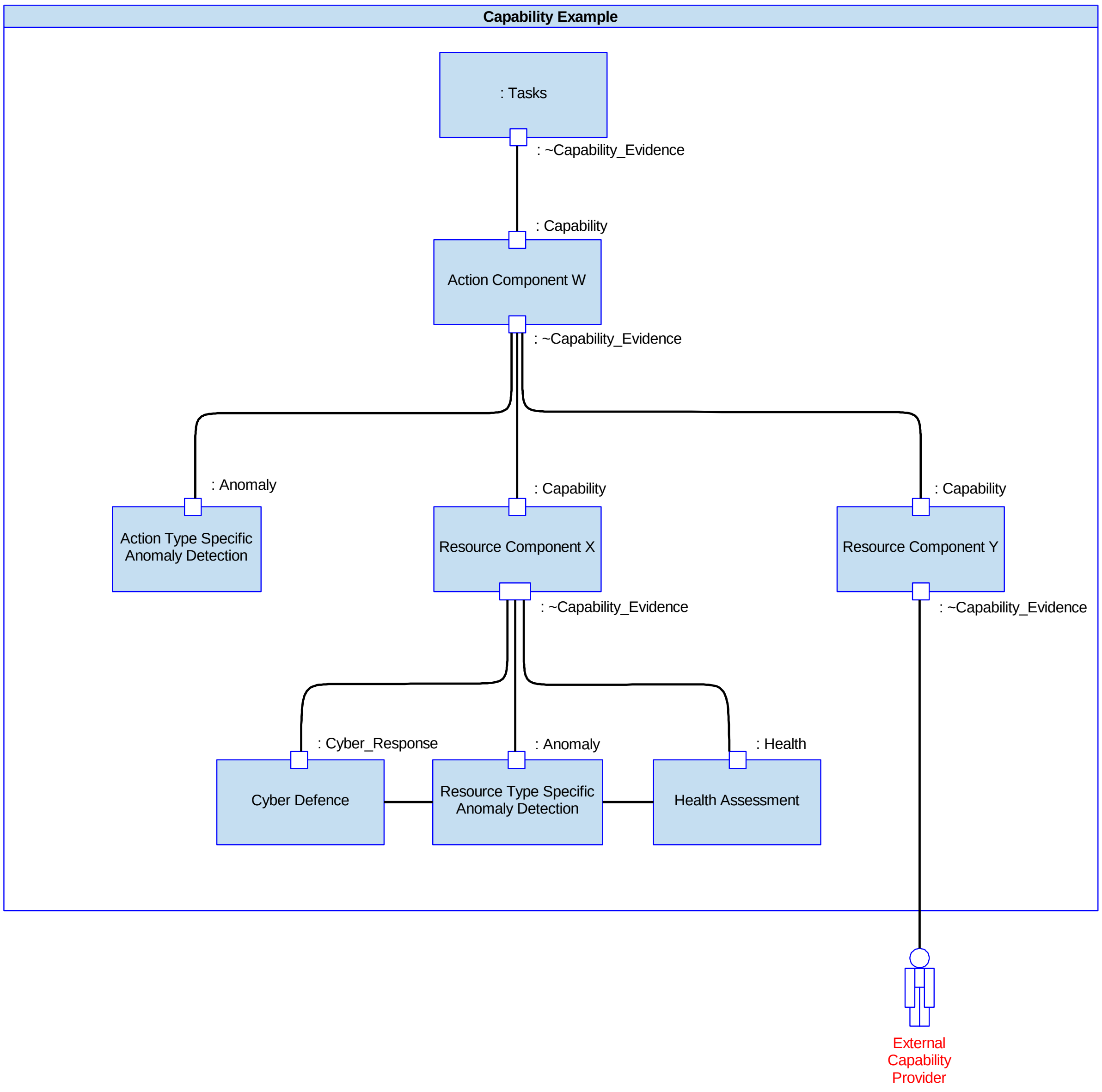


Figure 48: Capability Example: Structure

Resource Component X can receive capability evidence from Resource Type Specific Anomaly Detection, Health Assessment and Cyber Defence. Instances of these components are created to provide capability evidence specific to the resource. This could be:

- From Resource Type Specific Anomaly Detection: an anomaly that maps directly onto a capability. For safety-critical resources, an error message often identifies the loss of a capability unambiguously.
- From Health Assessment: a change in the health status of the resource which directly indicates the loss of a capability.
- From Cyber Defence: a change in the threat status of the resource because it has been compromised by a cyber attack and should be ignored.

Resource Component Y receives capability evidence directly from its resource, which is shown as an actor external to the system.

Also shown is Action Type Specific Anomaly Detection which is tailored to identify anomalies that only become apparent at the action level when information from different parts of the system is compared.

There will also be other interactions between these components as they cooperate to plan and execute a task, but these are not shown.

In this example, Health Assessment and Cyber Defence provide capability evidence only to Resource Component X, but they can and should provide capability evidence to all components, not just resource components. This is because Health Assessment and Cyber Defence are able to provide information about the underlying computing infrastructure, which is relevant to all components' capability evidence. This may also require a component to resolve differences in the capability evidence provided by different sources about the same capability, such as where the infrastructure is healthy but is unavailable due to potentially being compromised by a cyber attack.

A.2.6.5.1.2 Behaviour

Figure 49: Capability Example: Behaviour shows the behaviour associated with the example above. It shows how the components (represented by swimlanes) behave when a capability change occurs. As Resource Component Y has no capability or evidence change, its behaviour is irrelevant and has been omitted for this example.

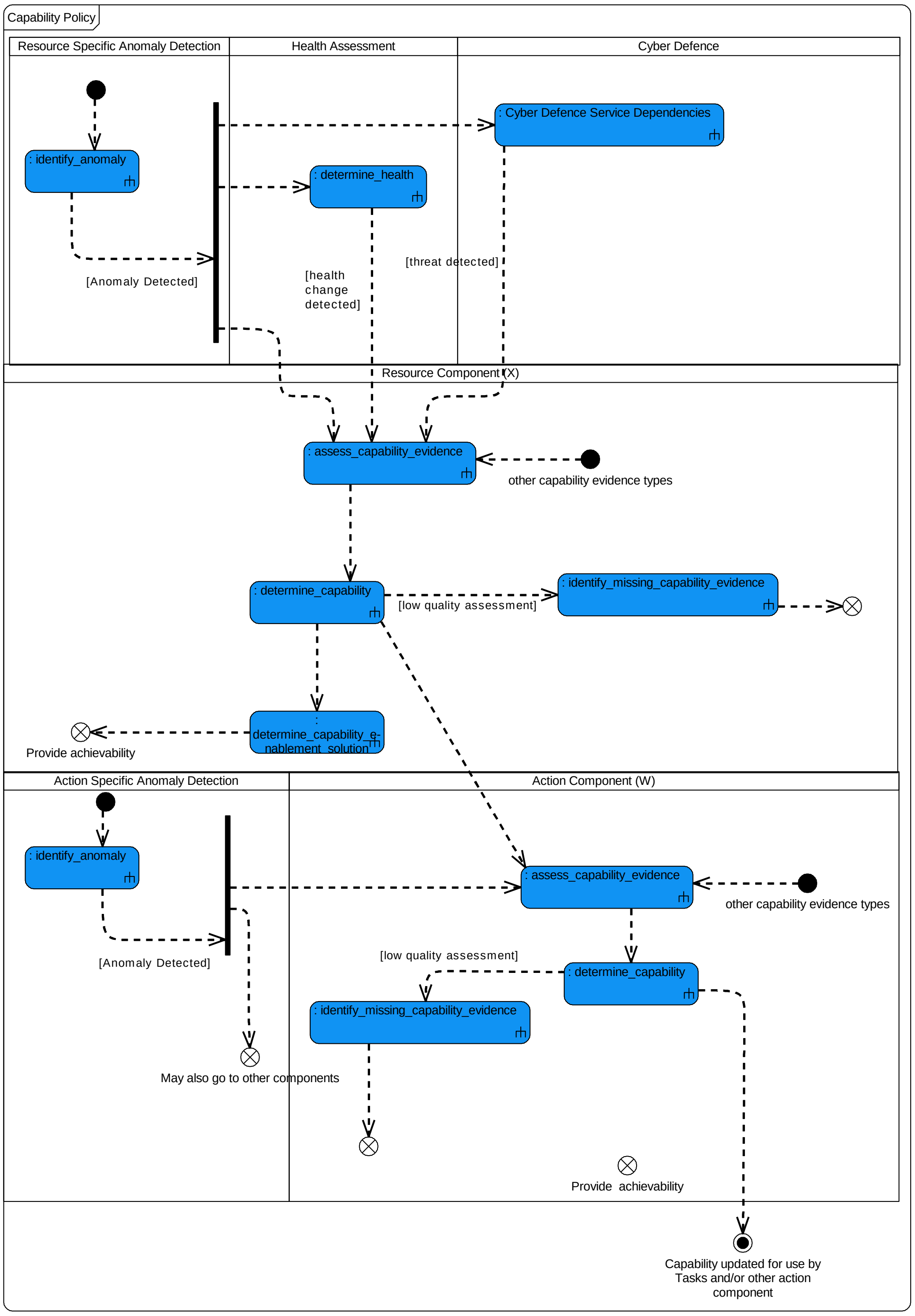

Capability Policy
Resource Specific Anomaly Detection
Health Assessment
Cyber Defence
: identify_anomaly
[Anomaly Detected]
: determine_health
[health change detected]
: Cyber Defence Service Dependencies
[threat detected]
Resource Component (X)
: assess_capability_evidence
other capability evidence types
: determine_capability
[low quality assessment]
: identify_missing_capability_evidence
: determine_capability_enablement_solution
Provide achievability
Action Specific Anomaly Detection
Action Component (W)
: identify_anomaly
[Anomaly Detected]
May also go to other components
: assess_capability_evidence
other capability evidence types
[low quality assessment]
: determine_capability
: identify_missing_capability_evidence
Provide achievability
Capability updated for use by Tasks and/or other action component

Figure 49: Capability Example: Behaviour

When a problem occurs, Resource Type Specific Anomaly Detection will identify an anomaly and notify Resource Component X, Health Assessment and Cyber Defence (For more detail, see Health Management PYRAMID concept). If the anomaly can be mapped to a capability change, Resource Component X will recognise it at the Capability_Evidence service and will react to it by re-assessing its capability and re-considering the feasibility of current or planned activities. Resource Component X updates its Capability service to inform Action Component W (and other components that can use its capability) of the change.

Action Component W re-assesses its own capability based on the new capability evidence supplied by Resource Component X. As with all components, it uses the Capability_Dependency_Mapping (see Component Composition) to make the assessment. This captures the relationship between a component's own capabilities (expressed in terms of its own subject matter) and the capability evidence supplied to it. Not all changes in capability evidence result in a change to the overall capability of a component, if it has other resources available to it.

In the example there is a change, however, which Action Component W reports to Tasks using its Capability service. In this way the change in capability of Resource Component X is communicated, at an appropriate level of abstraction, to all components that need to know about it.

The feasibility of current or planned activities is covered in the Dependency Management PYRAMID concept.

In parallel, Health Assessment and Cyber Defence will assess the anomaly to decide whether it represents a health change or cyber threat. This may result in additional information about the capability change which may be more specific and allow the component to improve its assessments.

Action Type Specific Anomaly Detection identifies anomalies that arise from inconsistencies between different parts of the system that are only visible at the action level. Any such anomalies are reported to Action Component W (and also to action-level instances of Health Assessment and Cyber Defence components, not shown in this example).

A.2.6.5.2 Missing Information

If a capability assessment is not sufficiently specific or certain, the component may identify an item of capability evidence which could improve its assessment. Such information may be available only on demand (as a test result, for example). Since the requested information may not be available immediately (or at all) a component must not wait for the information to arrive, but should report the results of its current assessment. The component may also need to request improvements to the quality of the capability evidence received, for example due to a time delay between updates or to increase the confidence in the capability evidence. When (if) the additional information arrives, in the form of capability evidence, it may result in an improved capability assessment, the result of which is provided on the Capability service in the usual way.

A.2.6.6 Capability Shortfalls and Recovery

A capability shortfall occurs when the capability of a system has been degraded or lost altogether, resulting in the potential inability of the system to perform its required behaviour.

This section considers, once a capability shortfall has been identified, how the cause of the loss can be determined, as well as determining whether a resolution is required and if so how a resolution can be achieved. Whilst the resolution may be establishing a capability that has not been present from system start-up, for convenience, this section often describes resolutions in terms of capability recovery, though there is no notable difference between the two cases.

Within a system, there may be different levels of capability resolution ranging from low to high:

- **Low level:** involving a single component **and** with no consideration of other factors affecting whether capability should be restored and how it should be restored; to
- **High level:** involving multiple components **and** considering whether capability needs to be restored, balancing this need with wider considerations (such as the need for survivability and the need to meet mission objectives), and considering the best way to restore capability to have minimal impact on these wider considerations.

These two positions represent different ends of a spectrum, with various permutations in between.

It should be noted that the identification and resolution of capability shortfalls related to the execution platform that would normally be handled by the execution platform itself (i.e. operating system) is not the subject of this PYRAMID concept or within the scope of the PRA, other than being able to interact with and react to execution platform instructions.

A.2.6.6.1 Determining Loss of Capability

Sufficiently understanding why capability has been lost or is missing is often a necessary first step towards restoring the capability.

A component can obtain relevant information about the cause of a capability shortfall through its Capability_Evidence service. This will generally involve similar interactions to those used to determine the availability of capabilities that it relies on, as described previously in this PYRAMID concept. The Health Management and Cyber Defence PYRAMID concepts provide further information about determining the best known route cause of system hardware health and cyber-attack issues.

A.2.6.6.2 Resolving Missing Capability

A capability shortfall may not always result in the need to restore the capability. For example, the lost capability may not be required to fulfil the current requirements on the system, and so restoring the capability may unnecessarily use system resources and may be to the detriment of performance against achieving the requirements that have been placed on the system.

A.2.6.6.3 Low Level Capability Restoration

There will be many cases where an individual component is able to determine if capability recovery is required and how the recovery should be achieved. An example of this is where a resource component needs to trigger a restart of the equipment under its control due to a degradation in performance or a failure. In this example it is assumed that the resource component has access to any necessary information informing it of any constraints that should prevent it from automatically triggering the restart, such as if equipment is being used for a critical purpose where the degradation is acceptable.

A.2.6.6.4 High Level Capability Restoration

In many cases, the determination of whether capability should be recovered may be performed by a different component to the component that is reporting the capability shortfall. Furthermore, the recovery of the capability may involve multiple components, and potentially may not involve the component that reported the capability shortfall. The decision to recover capability or the solution to doing so may need to consider factors such as the relative importance of survivability verses the mission goals. This therefore has the potential to involve components in the higher layers of the control architecture, such as the Tasks component, rather than just action and resource components.

Some capability shortfalls may be the indirect result of deliberate actions of the system or operator. For example, an actuator may become ineffective due to freezing, where the altitude of the air vehicle is a significant contributing factor to the freezing. In such cases high level interventions, such as changing the air vehicle altitude (and therefore the surrounding air temperature), may be appropriate but may also need to be weighed against other factors relevant to the mission. Of course, consideration should be given during the Exploiting Platform design process about whether it is acceptable for the system to deliberately perform actions that could result in such situations in the first place, especially where safety and permanent equipment degradation are concerned.

Some solutions to capability recovery may result in the degradation of other capabilities and so, again, require the relevant component to determine if this is acceptable. For example, damage to the air vehicle may require stores to be jettisoned in order to regain the ability to perform certain manoeuvres. The jettisoning of stores will result in a loss of capability, such as air vehicle range, if fuel drop tanks are jettisoned, or self defence capability, if weapons are jettisoned. The appropriate decision therefore needs to be made about which capability is most important, which may have mission and safety implications.

A.2.6.7 Capability Management Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Capability Management Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept identifies the following generic component behaviours in relation to capability management: • Individual components are responsible for assessing their own capability. Components report on the status of their own capability and utilise capability evidence from dependents. • Individual components are responsible for identifying missing information which could improve the certainty or specificity of the capability assessment. • Individual components are responsible for predicting the progression of their capability over time and with use. • Individual components are responsible for addressing shortfalls when their capabilities are unavailable or are degraded.	The PRA defines the following generic component responsibilities and services in relation to capability management: **Responsibilities:** • identify_missing_information • assess_capability • predict_capability_progression • address_capability_issue **Services:** • Capability • Capability_Evidence
Specific PRA Component support for the PYRAMID Concept	The PRA separates the following areas of functionality that relate to capability management: • The detection of anomalies that might indicate a system health problem. • The assessment of the current and projected health of system hardware items. • The identification of, and response to, cyber attacks.	The Capability Management policy defines the following components whose roles relate to health management and cyber defence: • Anomaly Detection • Health Assessment • Cyber Defence For more information about these components, see the Health Management and Cyber Defence PYRAMID concepts.
Exploiter considerations	This PYRAMID concept does not identify any specific Exploiting Programme considerations.	N/A

Table 7: Capability Management PYRAMID Concept Summary

A.2.7 Multi-Vehicle Coordination

A.2.7.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

- Control Architecture

Read in conjunction with

- Dependency Management
- Capability Management
- Resource Management
- Component Connections
- Constraint Management

A.2.7.2 Introduction and Scope

This PYRAMID concept presents the general principles of multi-vehicle coordination, how multi-vehicle coordination can be used, and how its application is supported at different layers in the PRA Control Architecture.

A.2.7.2.1 What is Multi-Vehicle Coordination?

Multi-vehicle coordination provides the facility to plan, control and execute activities in a way that pools the capabilities of all the vehicles within a cooperating group (such as a flight of aircraft) in order to achieve a greater operational effectiveness than if the same vehicles were deployed to operate independently from each other. An operational example of this would be where weapon deployment and target illumination are provided by separate vehicles, which are both being tasked using intelligence from a third, stand-off vehicle. The membership of a cooperating group is likely to be dynamic, with vehicles operating independently for some parts of a mission (e.g. for aircraft take-off and landing phases) and joining together to cooperate on specific mission activities. Even whilst part of a cooperating group, some vehicle functions will remain independent, for example the control of each aircraft's flightpath. Hence the system functionality of each vehicle in a cooperating group is split into that which remains part of a single vehicle system context, and that which is included in the cooperating multi-vehicle system context.

In PRA terms, support to the operational use of multi-vehicle coordination stems from the following principles:

- The PRA does not limit the scope of a system of interest to a single vehicle or node - it is equally applicable to a system containing multiple vehicles or nodes.
- Multiple instances or variants of PRA components can be deployed on different vehicles or nodes within a multi-vehicle system context. See the Multiple Component Instances and Variants section of this PYRAMID concept for further details.
- PRA control and coordination principles are fully scalable and can accommodate the physical partitioning of systems at any level, including for the coordination of multiple vehicles.

A.2.7.2.2 Why is Multi-Vehicle Coordination Important?

The ability to use multiple vehicles to achieve mission objectives is an important operational consideration as it can:

- Enable the use of specialised or single role vehicles which can be better optimised to their intended use.
- Promote mission redundancy, as the loss of a single vehicle can be mitigated by other vehicles.
- Mitigate risk to high value/crewed vehicles by using lower value/uncrewed vehicles to perform high risk activities.
- Enable (or increase the effectiveness of) certain types of mission activities.

The PRA is designed to support coordination between different vehicles under various circumstances. This enables different coordination approaches to be adopted and different coordination behaviour employed on different types of mission. It gives PRA exploiters the ability to utilise multi-vehicle coordination in a flexible way without significant system modification.

When coordinating multiple vehicles an Exploiting Platform must take into account emergent issues such as latency. Although these issues are not addressed in the PRA, it does allow the necessary interactions to enable platform specific models to address them, such as the synchronisation interactions used in Figure 55: Adaptive Coordination Multi-Vehicle Actions Example.

A.2.7.2.3 Exclusions to this PYRAMID Concept

The following areas of functionality related to multi-vehicle coordination are provided within the PRA but are not discussed by this PYRAMID concept:

- Uncrewed Air Vehicle (UAV) control handover.
- Operator interaction and control (including autonomy).

While the act of coordination may include elements of control, this PYRAMID concept excludes the concepts and methods of control outside that of coordination between aircraft. Hence the concepts and achievement of control including operator control, ground control nodes, etc., are not covered.

A.2.7.3 Overview

This PYRAMID concept considers the concepts of coordination between vehicles in a number of ways - this may be to allow multiple vehicles to achieve either a common goal or individual goals, and can involve the whole capability of a vehicle or part of it. Coordination can be achieved through two types of interaction between vehicles:

- Instruction
- Shared Information

The main consideration relating to the deployment of the PRA in a multi-vehicle context is the use of multiple instances or variants of PYRAMID components across the different vehicles in a cooperating group. Generally the PRA is written from a logical viewpoint and does not consider system partitioning, however the flexibility and scalability of the PRA means that its principles and PYRAMID concepts remain valid once applied to a physical architecture where multiple instances of the same PYRAMID component are required. This PYRAMID concept describes how relevant PYRAMID concepts can be applied in a multi-vehicle context, most importantly:

- Control Architecture
- Capability Management
- Constraint Management
- Dependency Management

Furthermore, the PYRAMID concept outlines how the Flights and Formations components can aid multi-vehicle coordination by managing the composition of a flight and coordinating the relative position of vehicles.

A.2.7.4 Multi-Vehicle Coordination Concepts

A.2.7.4.1 Types of Coordination

Coordination is the organisation of the different elements of a complex body so as to enable them to work together effectively. In the context of this PYRAMID concept this covers both the:

- Coordination of 2 or more vehicles operating together to achieve a common goal.
- Coordination of 2 or more vehicles operating together to achieve their individual goals.

Such coordination may be applied at either:

- A whole vehicle level.
- A part of the overall capabilities of a coordinated vehicle.
- A combination of whole vehicle and part vehicle capabilities.

The coordination of a system and its parts (including its components) is achieved by two types of interaction: instruction and shared information. Within the context of this PYRAMID concept these terms are defined as follows:

- **Instruction** is a request from one part of a cooperative system to another to achieve an activity on its behalf, for example a control message or requirement.
- **Shared Information** is information contained within one part of a cooperative system that enables another part to act in a cooperative manner, or forms a response to a cooperative activity.

One or both of these interaction types will be required by the elements of a cooperative system in order to coordinate their activities.

A.2.7.4.2 Multiple Component Instances and Variants

The PYRAMID Reference Architecture is a platform independent logical architecture - by definition it does not specify how it will be deployed onto physical systems.

In contrast, the principles of multi-vehicle coordination are largely concerned with the physical aspects of a system - the system of interest in this case is a group of physically separated vehicles that will cooperate to provide the required system functionality (e.g. to meet mission objectives).

Using the PRA, a deployment specific logical architecture can be created that will meet a system's functional requirements by considering the subject matter of the PRA components and the PYRAMID concepts that underpin their use. This logical architecture makes no reference to how the deployed system is physically partitioned - it should be valid regardless of the physical partitioning of the system. This view is illustrated in Figure 50: System Logical View.

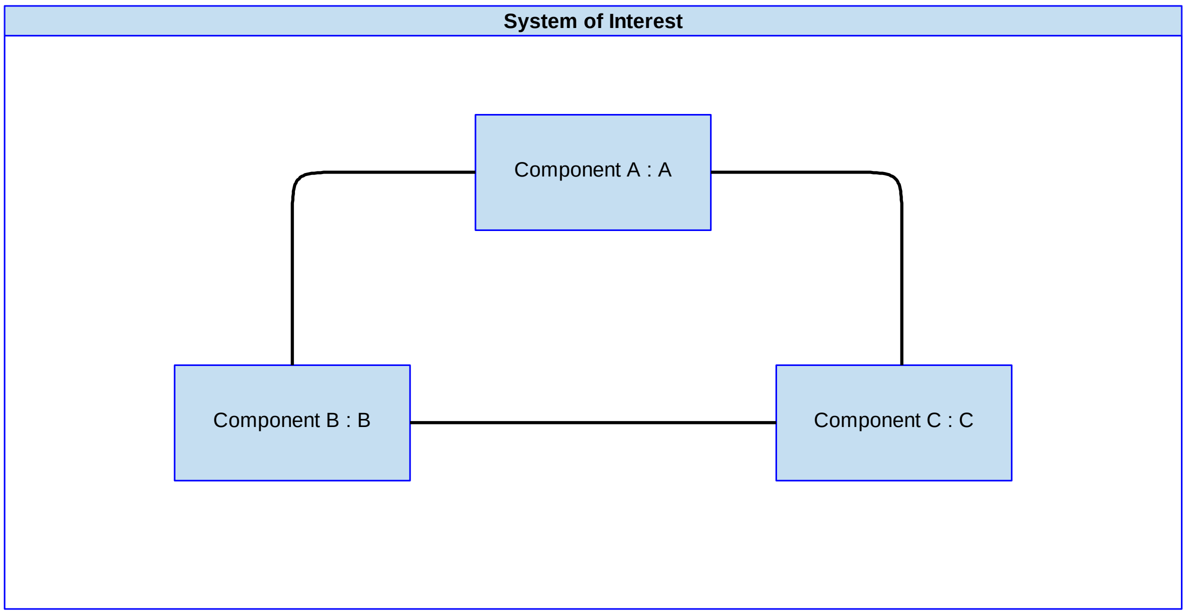


Figure 50: System Logical View

If there is a requirement for the deployed system to support multi-vehicle coordination, there will be a need for some partitioning of system functionality between vehicles. If this partitioning is at a whole component level (i.e. each component is allocated to a single vehicle), the physical system architecture remains equivalent to the logical system architecture. Deployment-specific aspects of the system will need to be considered, for instance the addition of components to handle communications between the different vehicles and non-functional aspects such as redundancy and latency, but these are out of the scope of this PYRAMID concept. In functional terms, the interactions between components A, B and C do not change due to the physical partitioning of the deployed system. This is illustrated in Figure 51: Partitioned System View - Equivalent to Logical View.

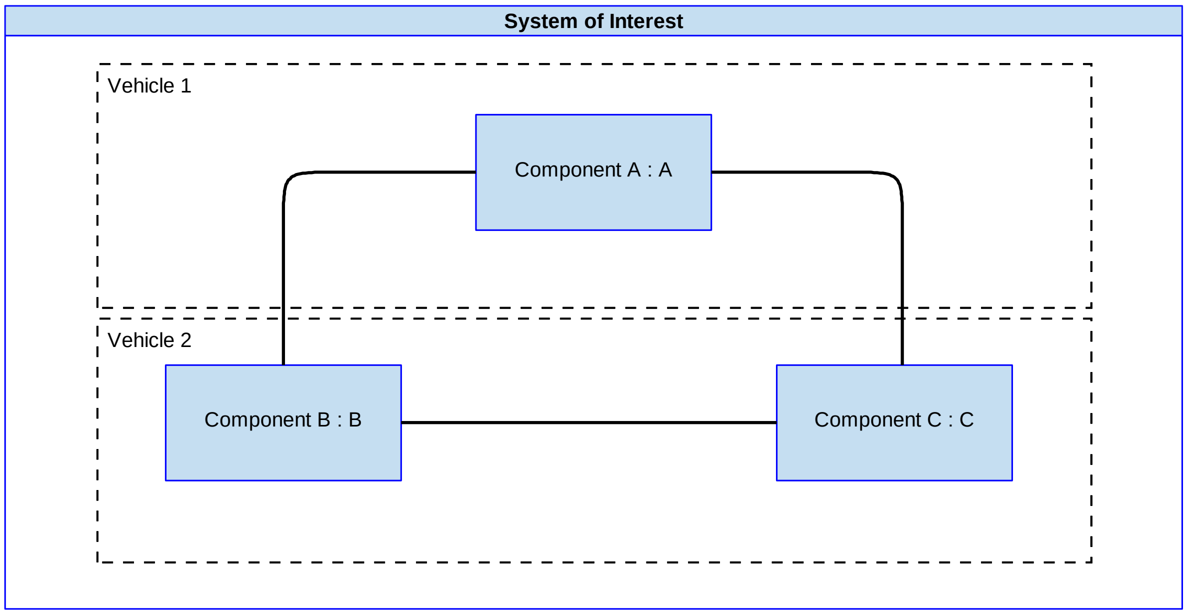


Figure 51: Partitioned System View - Equivalent to Logical View

However, if the system partitioning is below a whole component level (i.e. the functionality provided by a single PRA component needs to be split or replicated across vehicles), this will result in a physical system architecture that is no longer equivalent to the logical system architecture. In the example being used, the functionality of Component C could be provided by two cooperating vehicles, as illustrated in Figure 52: Partitioned System View - Not Equivalent to Logical View. Whilst the functionality of the system of interest has not changed from that shown in Figure 51: Partitioned System View - Equivalent to Logical View, the physical scope of the system has increased to include Vehicle 3.

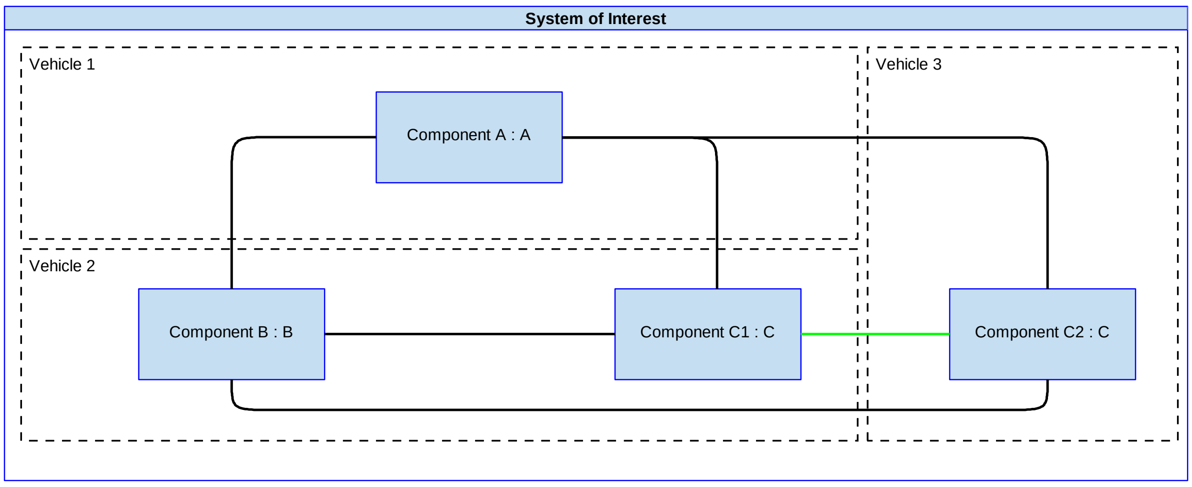


Figure 52: Partitioned System View - Not Equivalent to Logical View

Any decision about when or how to use multiple instances or variants of PRA components is exclusively a deployment consideration - it is neither discouraged nor recommended by the PRA. Nevertheless it is a fundamental principle that supports the partitioning of deployed systems at any level, including in a multi-vehicle context. In Figure 52: Partitioned System View - Not Equivalent to Logical View, the component C (C1 and C2) can be either:

- **Functionally identical** - where component C1 and component C2 provide identical functionality. Examples of this kind of use would be for redundancy purposes, where one version of the component is acting as the primary instance and providing the required functionality to the system. The other instance would not be providing the same functionality concurrently, though it may be synchronising with the primary instance to allow it to take over as the primary instance if and when required.
- **Functionally different** - where component C1 and component C2 are providing different functionality, e.g. the responsibilities of component C are distributed between the two variants (though they may also share some of the same responsibilities) or two instances with different data driven runtime data.

Again, the use of different PYRAMID component instances and variants will depend on deployment considerations and is not considered within the PRA. Components C1 and C2 must however follow the normal rules of PYRAMID compliance, that is their subject matter must only be that of PRA component C and not of any other PRA component (though they can be different from one another). The interactions between components C1 and C2 are treated as internal interactions from a PRA point of view, as indicated by the green line in the diagram, hence are not defined as PRA services and will be deployment-specific (as mentioned in PYRAMID Technical Standard, Ref. [1], section Component Services Not Defined by the PRA).

It can be seen from Figure 52: Partitioned System View - Not Equivalent to Logical View that the replacement of component C with components C1 and C2 has increased the number (and potentially changed the type) of interactions within the system of interest. This PYRAMID concept will demonstrate how the principles and concepts of PYRAMID, written largely from a logical viewpoint, can be applied when deploying multiple instances or variants of components in order to handle these extra interactions and hence support the operational application of multi-vehicle coordination.

A.2.7.5 PRA Application of Multi-Vehicle Coordination

A.2.7.5.1 Multi-Vehicle Control Architecture

A.2.7.5.1.1 Control Architecture Principles

As stated in the Control Architecture PYRAMID concept:

"The control architecture can span multiple systems, including multiple vehicles, to enable integrated or compatible systems that are capable of achieving interoperable, coordinated, and holistic solutions. This can require coordination between different instances of the same PYRAMID component or between different PYRAMID components within different systems."

This means that the principles detailed in the Control Architecture PYRAMID concept are equally applicable in a multi-vehicle context. In particular:

- The control architecture layers extend across all the vehicles in a cooperating group.
- A rigid control hierarchy is not imposed by the PRA across a cooperating group of vehicles.
- Delegation of requirements through the layers of the control architecture (i.e. coordination through instruction) may remain the same regardless of the separation of components across

different vehicles. For example, a Communication Links component could place requirements on a Communicator component on another vehicle for coordination of a communications action. However, in some cases the control architecture may need to allow for limitations and safeguards to prevent a component on one vehicle from placing requirements on another vehicle, in order to cater for cases where a cooperating group is only under partial control of a controlling vehicle.

- Coordination between components through shared information can also occur regardless of the separation of components across different vehicles. For example, a Tasks component can exchange information with another version of the Tasks component on a different vehicle for coordination of inter-vehicle task demands.

As described in the Multiple Component Instances and Variants section of this PYRAMID concept, extra complexity is added as a result of the use of multiple versions of components across a cooperating group of vehicles, and in the context of the control architecture, this is particularly relevant to the use of the Objectives and Tasks components.

The Control Architecture PYRAMID concept states that the Objective and Task Layers each contain a single component (Objectives and Tasks respectively), and whilst this is true in a logical architecture context, in a deployed multi-vehicle system context it is possible (and acceptable from a PRA perspective) that each vehicle in the group will contain instances or versions of one or both of these components. These components may have to function under different control structures during a mission, for instance:

- Whilst not part of a cooperating group. In this case, the control architecture scope is that of the single vehicle.
- Whilst part of a cooperating group under full control of a controlling vehicle.
- Whilst part of a cooperating group under partial control of a controlling vehicle. In this case some capabilities of all vehicles within the cooperating group can be controlled by the controlling vehicle, and some can only be controlled by components which are located on the same vehicle. This is the most likely option for aircraft operating in a flight, where aircraft joining the flight are unlikely to surrender control of all their functionality to the flight lead.

Such considerations are specific to the requirements of a deployment, however the flexible nature of the Control Architecture PYRAMID concept will allow them to be catered for.

A.2.7.5.1.2 Multi-Vehicle Control Pattern Examples

Whilst the Control Architecture PYRAMID concept does not impose a rigid control hierarchy, the following examples of patterns of integration of components in different multi-vehicle control scenarios are provided to show how differing control interactions can be used (as required) at differing levels of a system’s capability:

- **Objective coordination** is coordination of planning at the Objective layer, where different vehicles either coordinate between versions of Objectives components, or are assigned distinct tasks that collectively deliver an overall mission objective. This is further described in Objective Coordination.

- **Task coordination** addresses coordination of planning at the Task layer, where different vehicles either coordinate between versions of Tasks components, or the vehicle responsible for task coordination provides Tasks a sequence of actions which may then be assigned to different vehicles that can collectively contribute to a task. This is further described in Task Coordination.
- **Action coordination** addresses coordination of execution at the Action and Resource layers through the flexible distribution (across multiple vehicles) of synchronised activities that contribute to the same complex action. This can be based on a dynamic understanding of capability regardless of vehicle role, and may involve data sharing directly between Action and Resource Layer components. This is further described in Action Coordination.

A.2.7.5.1.2.1 Objective Coordination

The highest level of multi-vehicle coordination is carried out at the Objective layer, where different vehicles are assigned distinct tasks that collectively contribute to an overall mission objective. Each co-operating vehicle performs discrete tasks.

These tasks may be provided to the tasked vehicles by an Objectives component from another vehicle (e.g. a flight lead) or by a local instance of the Objectives component which is coordinated with other instances or variants of Objectives components on other vehicles.

For a cooperating group of vehicles under full control of a flight lead, the Objectives component on the flight lead will be aware of all capabilities of the cooperating vehicles. It will be able to place any tasks enabled by the capabilities of another vehicle on the Tasks component of that vehicle.

For a cooperating group of vehicles under partial control of a flight lead, the flight lead Objectives component should be aware of the capabilities of the cooperating vehicles which it is permitted to command. Any task placed directly on the Tasks component of another vehicle should be within the scope of those capabilities.

An example where this type of high-level coordination is appropriate would be a planned Suppression of Enemy Air Defences (SEAD) mission, where vehicles have pre-assigned roles. A particular vehicle may be assigned as a weapon deployment vehicle, another may be a reconnaissance or primary sensing vehicle and another may be allocated to the role of stand-off jamming.

(Note in the example no further components other than Tasks and Objectives are illustrated.)

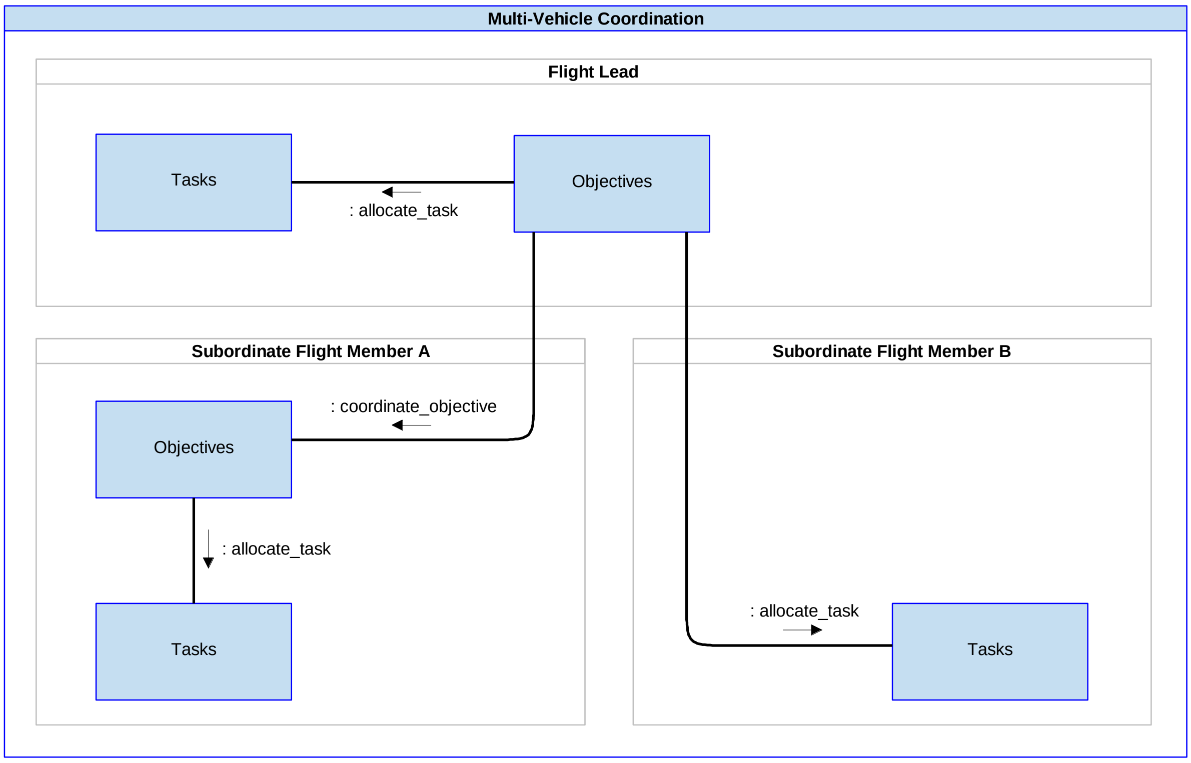


Figure 53: Objectives Coordinating Multi-Vehicle Tasks Example

A.2.7.5.1.2.2 Task Coordination

This level of multi-vehicle coordination occurs between versions of the Tasks component, and the Task and Action layers. Similar to the objective coordination example, coordinated Tasks components break down tasks into a series of actions allocated across the coordinated vehicles.

This allocation may be made on the basis of dynamic capability discovery as assessed and provided by the Action components from coordinating vehicles across the cooperating group.

In this type of coordination the Tasks component may be required to understand the capabilities of coordinated vehicles. For example, the capability of a flight member's Action components is communicated to a central coordinating vehicle, enabling use of knowledge of both single vehicle and multi-vehicle techniques that are available to achieve a task.

(This illustrates the sharing of commands and information between vehicles which achieves the coordination.)

The same principle of full or partial control which applies at the Objective Coordination level also applies at the task coordination level.

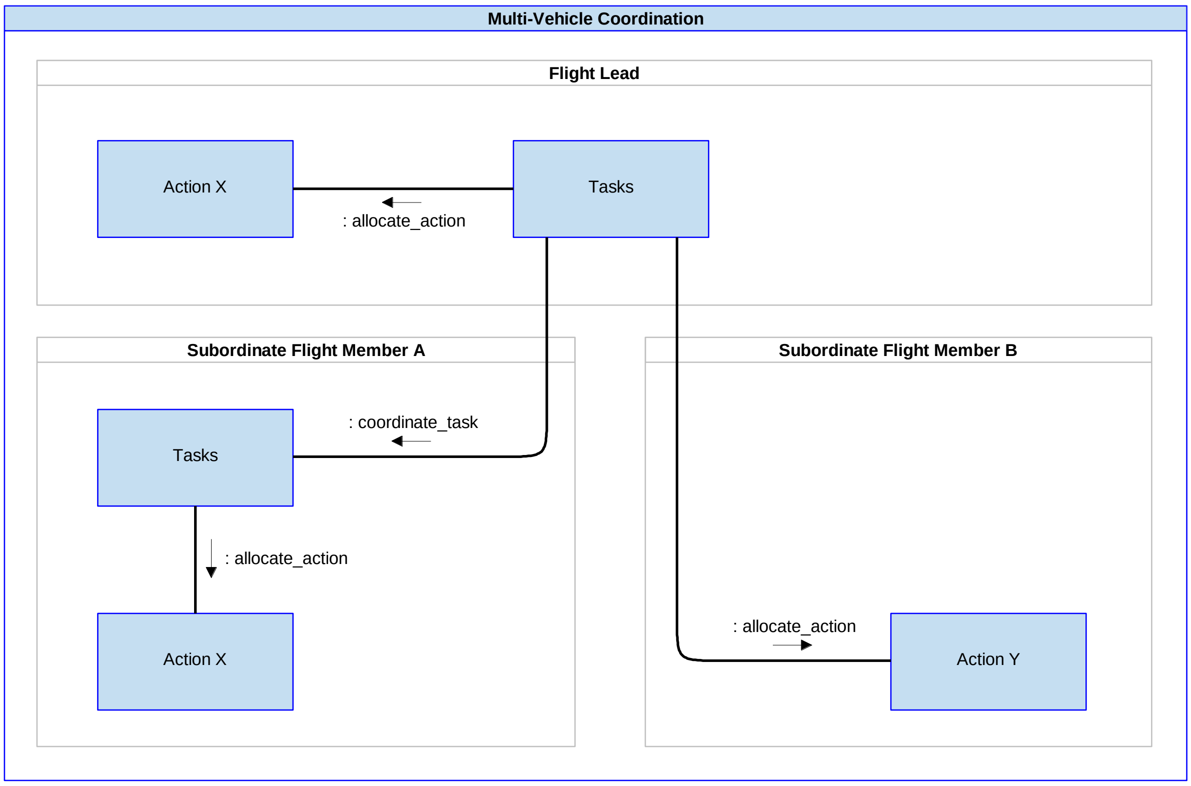


Figure 54: Tasks Coordinating Multi-Vehicle Actions Example

A.2.7.5.1.2.3 Action Coordination

Execution of some multi-vehicle actions requires a highly dynamic approach that is adaptive in nature. An example application that falls into this category is multi-vehicle passive geo-location, a technique that requires close coordination, synchronising and data sharing between different vehicles at the Action and Resource Layers. The relevant resource components on each participating vehicle must be able to synchronise detection of Electro-Magnetic (EM) energy and combine results for analysis.

Adaptive cooperation between vehicles at execution time requires PRA component interactions at the Action layer and below.

General principles of operation required at execution time for adaptive coordination of multi-vehicle actions are:

- **Coordination** between like Action layer components on different vehicles within a cooperating group.
- **Synchronisation and data sharing** directly between Resource layer components on different vehicles within the cooperating group.

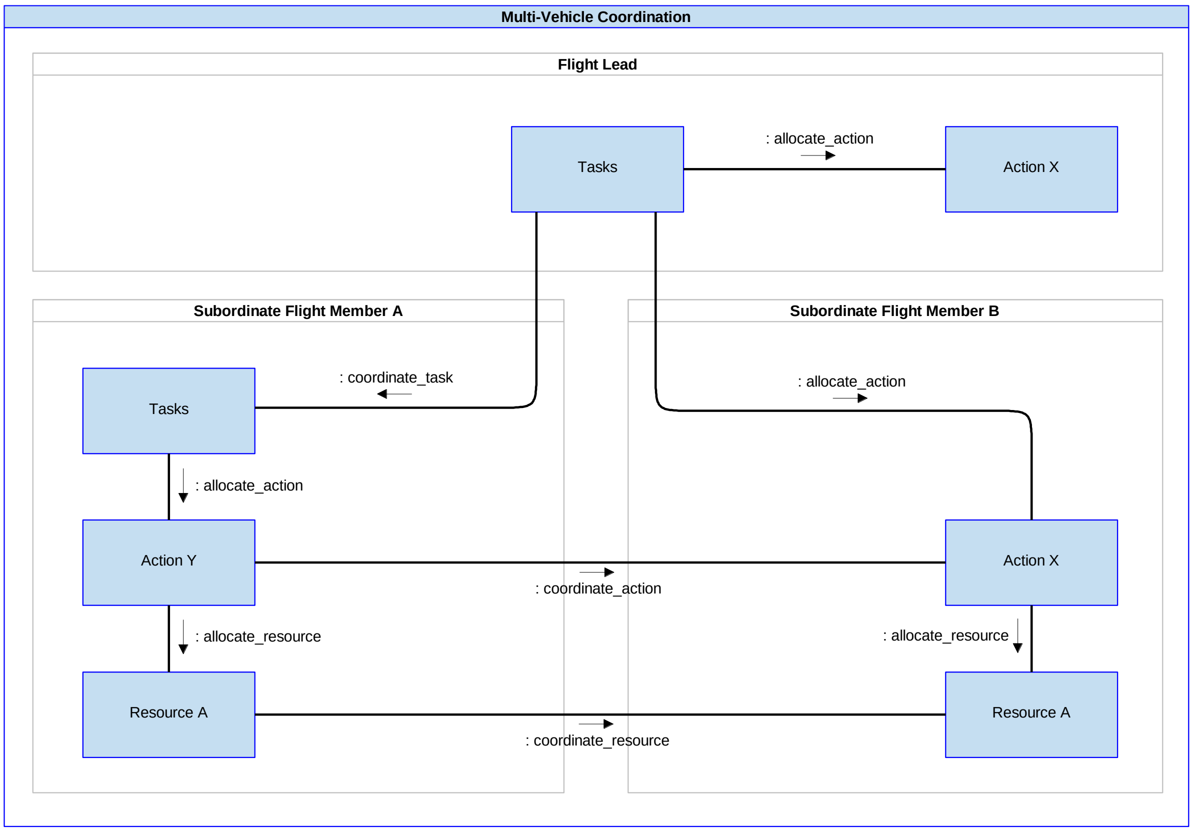


Figure 55: Adaptive Coordination Multi-Vehicle Actions Example

A.2.7.5.2 Multi-Vehicle Capability Management

The Capability Management PYRAMID concept states the important PRA principle that every component in the PRA is responsible for assessing its own capability, and this principle remains true for a system operating in a multi-vehicle context. Every component reports its capability to the components that make use of that capability, and this information is used in the capability assessment performed by those components. The capability reporting structure across a multi-vehicle system will be highly dependent on the control architecture used by that system (it generally includes capability reporting flows in the opposite sense to control), but the general PRA principle still applies - if the capability of a component on one vehicle is used by a component on another vehicle then the reporting of capability should be between those components.

The use of multiple instances or variants of a component does allow for different capability reporting mechanisms to be used, within the general principles of the Capability Management PYRAMID concept. Figure 56: Individual Capability Reporting shows the case where three instances or variants of the Routes component are reporting their individual capability to the Tasks component, which decides how to use those capabilities and places requirements on the three instances or variants of the Routes component accordingly across the three vehicles.

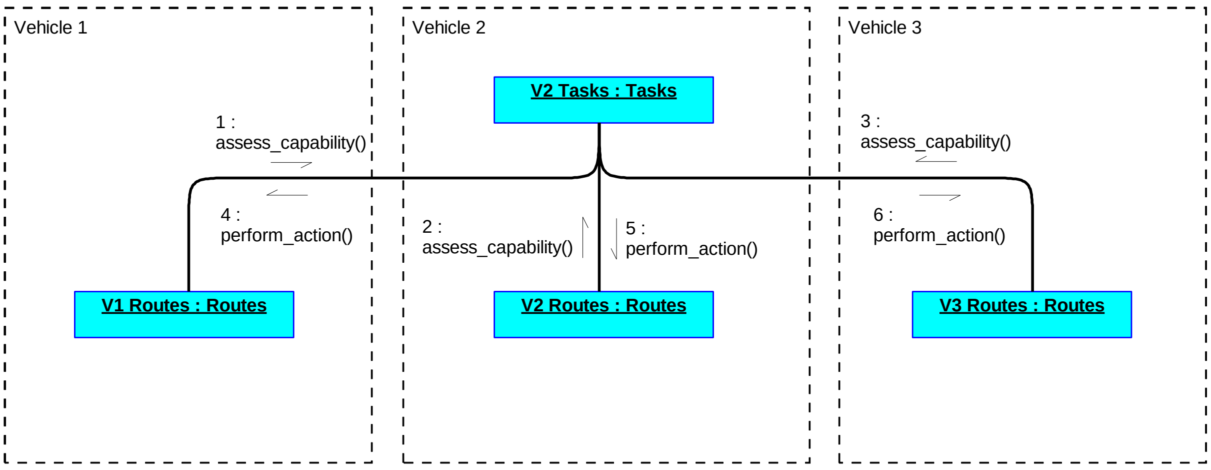


Figure 56: Individual Capability Reporting

Figure 57: Combined Capability Reporting shows the case where the two instances or variants of the Routes component in Vehicle 1 and Vehicle 3 report their individual capability to a third Routes component in Vehicle 2. The Routes component in Vehicle 2 provides a combined routing capability to the Tasks component, which decides how to use that capability and places requirements back on the Routes component in Vehicle 2. The Routes component in Vehicle 2 then decides how the three Routes components will coordinate to meet the requirements from the Tasks component (potentially using shared information across the three Routes components) and places requirements on the Routes components in Vehicles 1 and 3 accordingly. Any interactions between the three Routes components (indicated by the green lines on the diagram) are considered to be internal component interactions from a PRA point of view, hence they would not necessarily align to the services included in the PRA Routes component definition. This is described in PYRAMID Technical Standard, Ref. [1], section Component Services Not Defined by the PRA.

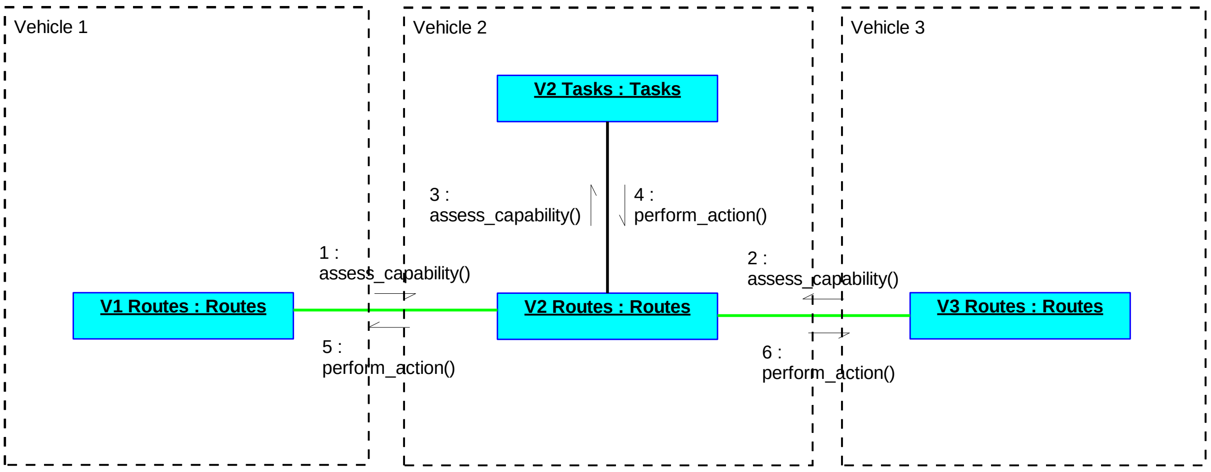


Figure 57: Combined Capability Reporting

A.2.7.5.3 Multi-Vehicle Constraint Management

The Constraint Management PYRAMID concept describes the principles relating to how system constraints can be managed across a system architecture using PYRAMID components. The PYRAMID concept defines three types of constraints (solution-based, internal, and rule-based) that can be used by a PYRAMID deployment. These principles extend to a multi-vehicle situation, allowing for the following:

- In principle any component can generate a constraint, regardless of which vehicle it is on.
- Constraints can be imposed by components on one vehicle onto components on another vehicle. This includes constraints from the Operational Rules and Limits component, which sets all rule-based constraints.
- Various options are available for where the Operational Rules and Limits component should reside, ranging from each vehicle having an Operational Rules and Limits component to a single vehicle having an Operational Rules and Limits component, which issues constraints to other vehicles. In the latter case the other vehicles may include an inactive Operational Rules and Limits component for redundancy purposes or to enable different configurations of vehicles within a flight.
- Regardless of which vehicles have an active Operational Rules and Limits component, the applicable rules may be different for different vehicles, or their associated limits may become active under different circumstances (e.g. due to different limits being applicable to different vehicle types or role fit configurations).
- The data sets used to data drive the Operational Rules and Limits component, or to data drive internal constraints for other components, may be different for their applicability to different vehicles.
- The contextual information used by a component to determine whether a constraint needs to be adhered to under specific circumstances could originate from the same vehicle or a different vehicle.

Whilst the PRA does not restrict the design choice for any of the above, the implications of such design choices should be carefully considered. For example, some design choices will be appropriate for crewed vehicles that control lower capability uncrewed vehicles, but the same design choices may be less appropriate for flights involving vehicles of equivalent type and capability, which have equivalent authority during a mission.

A.2.7.5.4 Multi-Vehicle Dependency Management

The Dependency Management PYRAMID concept describes the importance of managing dependencies between the components within a PYRAMID deployment in an explicit and flexible way. The general principles of the PYRAMID concept are:

- Components are responsible for identifying their own dependencies when generating solutions and placing requirements on other components to ensure those dependencies are satisfied.
- Components monitor the progress of their solutions based on achievement and progress reported by their dependent components; hence components also have a responsibility to report this information to components that have placed requirements on them.
- Planning context is used to maintain traceability between different solutions - this records how the involved components contribute to different solutions based on differing solution constraints.

In relation to a multi-vehicle context, the important point about the principles described in the Dependency Management PYRAMID concept is that they apply to dependencies between individual components, whether the components are situated on one vehicle or multiple vehicles. Hence, the PYRAMID concept can be scaled to consider any number of vehicles.

A.2.7.5.5 Flights

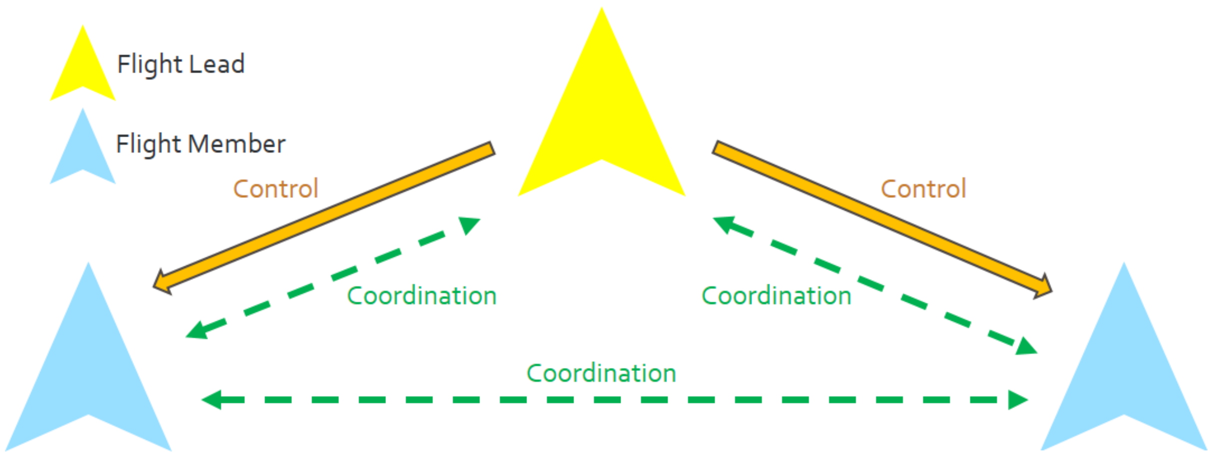


Figure 58: Example portraying Control and Coordination in a simple flight

A flight is a collection of aircraft working collaboratively to achieve mission objectives.

The concept of a flight supports a hierarchical approach to the control of different vehicles, with the flight lead being responsible for receiving the goals that the flight should achieve (such as objective or tasks) and assigning activities to flight members within the flight to achieve these goals.

As shown in Figure 58: Example portraying Control and Coordination in a simple flight, the flight hierarchy does not prevent information sharing and coordination activities from being carried out directly between flight members.

The Flights component is responsible for the management of the composition of a flight of air vehicles, including the assignment of different roles within the flight, such as flight lead and flight member. This includes, management of vehicles joining and leaving a flight, the handover of roles, and the merging and separation of entire flights, regardless of whether these are pre-planned or unplanned.

A.2.7.5.6 Formations

The Formations component is responsible for the spatial arrangement of a group of vehicles.

There is no inherent relationship between flight and formation membership. For example, a formation may include vehicles which are not part of a flight, of which only a subset are part of a flight, or belong to multiple flight.

The role of the Formations component extends to the consideration of vehicles beyond its control, where the position of a controllable vehicle relative to a non-controllable vehicle needs to be managed.

To fulfil its responsibilities, Formations can:

- Manage members of the formation, including adding or removing formation member.
- Identify a vehicle, or vehicles, that other vehicles are positioned relative to.
- Coordinate the spatial arrangement and movement of vehicles within the formation.

The subject matter of Formations therefore includes knowledge of formation patterns and movement sequencing. Some of this information may be data-driven within the component.

A.2.7.5.7 Other Multi-Vehicle Considerations

The preceding sections outline the concepts of multi-vehicle coordination and the most important PYRAMID principles and concepts that support it. This section indicates other PYRAMID concepts and interaction views (IVs) that, in addition to those stated in the Pre-Reading and Related PYRAMID Concepts section, may be of use in support of these concepts.

A.2.7.5.7.1 Communication between Vehicles

For guidance on the application of communications between vehicles / nodes using the PRA, refer to the Use of Communications PYRAMID concept and Appendix C: Interaction Views, Communications Views IVs, especially Figure 166: Connection Management IV.

A.2.7.5.7.2 Time Synchronisation between Vehicles

Where a multi-vehicle coordinated activity requires vehicles to be synchronised for their achievement, Figure 189: Time Synchronisation between Nodes IV can be consulted. This would be supportive of examples such as the coordinated use of distributed sensor technologies such as shown in Figure 96: Radar and EO System Example.

A.2.7.6 Multi-Vehicle Coordination Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Multi-Vehicle Coordination Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept does not identify any generic component behaviours in relation to multi-vehicle coordination. Nor does the PYRAMID concept mandate any specific approach to achieving multi-vehicle coordination. However, all PRA components support the following approaches to multi-vehicle coordination (or combinations thereof) to enable the passing of instructions, or the sharing of information, between vehicles: • Interaction between different PRA components instantiated on different vehicles, e.g. for the purpose of distributed tasking or resource usage. • Interaction between different variants of the same PRA component instantiated on different vehicles. • Interaction between different instances of the same PRA component instantiated on different vehicles.	The PRA principles outlined in the Control Architecture, Capability Management, Constraint Management, and Dependency Management PYRAMID concepts apply in a multi-vehicle coordination scenario. The PRA does not define any generic component responsibilities in relation to multi-vehicle coordination. Component services to support the interaction between multiple instances or variants of a component are considered internal to the PRA component and are therefore not defined by the PRA.
Specific PRA Component support for the PYRAMID Concept	The PRA separates the following areas of functionality that relate to multi-vehicle coordination: • The management of the composition of a flight of air vehicles. • The coordination of the relative positions of vehicles moving in a formation. • Communication between vehicles.	The PRA defines the following components whose roles relate to multi-vehicle coordination: • Flights • Formations The PRA defines a number of components that deal with different aspects of communication. For more information see the Use of Communications PYRAMID concept.
Exploiter considerations	The following multi-vehicle coordination related considerations are a matter for individual Exploiting Programmes: • The specific approach to system partitioning to enable vehicle coordination, for example, the use of component variants and instances, as noted above. • The definition of internal component services to support interaction between variants or instances of a component. • Provision for handling communications latency or ensuring functional redundancy in a multi-vehicle coordination scenario.	The PRA does not mandate a specific approach in respect of these considerations.

Table 8: Multi-Vehicle Coordination PYRAMID Concept Summary

A.2.8 Interaction with Equipment

A.2.8.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Resource Management
- Control Architecture

A.2.8.2 Introduction and Scope

This PYRAMID concept explains how PRA components can be used to interact with external equipment. It describes how components throughout the architecture are involved and how the PRA accommodates interactions with the widest possible variety of independent equipment. It also explains how adding a new item of equipment to an Exploiting Platform can be managed.

The PRA allows for interfacing with equipment of any complexity level.

A.2.8.2.1 What is Equipment?

Equipment could be either hardware (for example, antennas or control surfaces) or a combination of hardware and software that provides a capability or resource to the system under consideration. The complexity of a specific piece of equipment could vary from a simple device such as a thermistor to complex equipment with multiple data interfaces, such as a Laser Designator Pod (LDP).

A.2.8.2.2 Why is Interaction with Equipment Important?

The PRA supports the design and configuration of a system capable of interacting with external equipment in order to meet its requirements. It gives PRA Exploiters the ability to incorporate legacy or new equipment into a system with minimal system modification.

A.2.8.3 Overview

The principles of a PYRAMID deployment interacting with equipment are:

- The PRA components used in a PYRAMID deployment to interact with equipment will be chosen to support the functional interfaces of the equipment.
- The interface of a PYRAMID deployment with an equipment could involve one or more PRA components.
- The components used in the interface can be from whatever layer of the Control Architecture is appropriate for the specific equipment interface. Figure 59: Component Interfaces at the PYRAMID Deployment Boundary shows an example.
- Some aspects of an equipment's interface may be supported by the execution platform rather than by PRA components.

A.2.8.4 Component Interfaces at the PYRAMID Deployment Boundary

The components used in the interface can be from whatever level of the Control Architecture is appropriate for the specific piece of external equipment. Figure 59: Component Interfaces at the PYRAMID Deployment Boundary illustrates this concept.

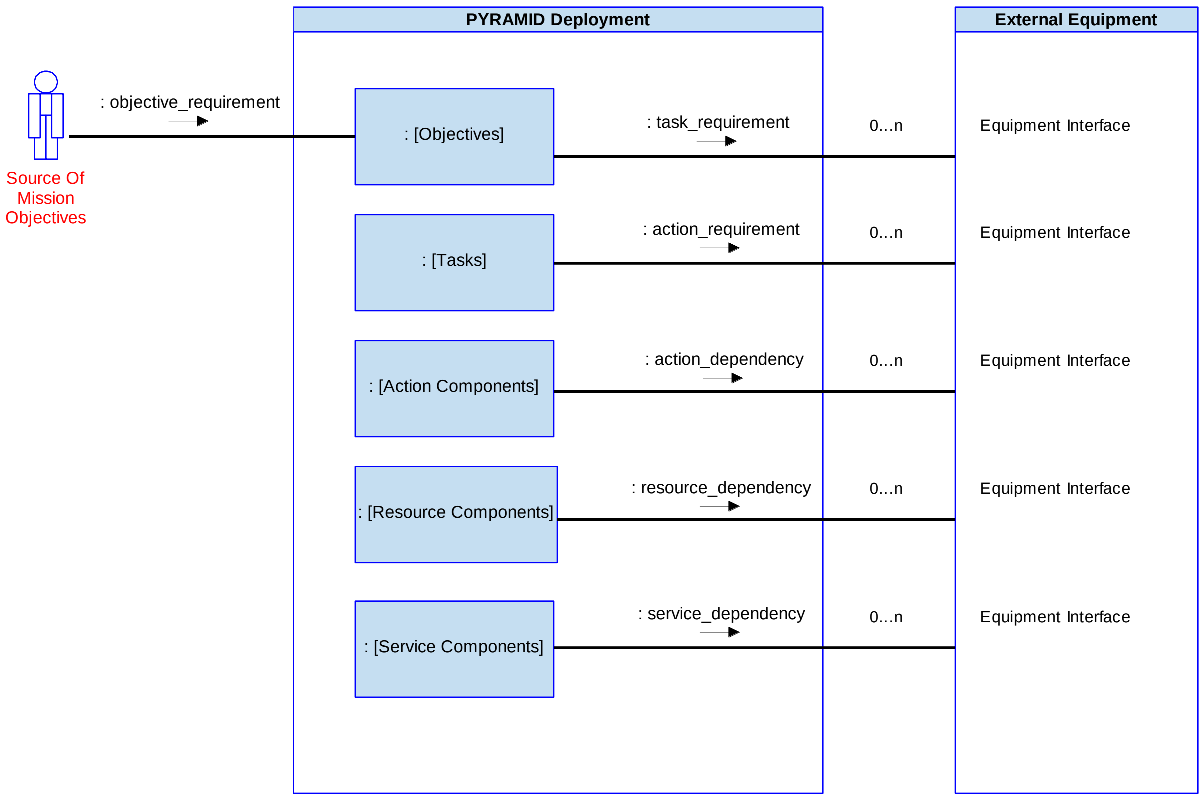


Figure 59: Component Interfaces at the PYRAMID Deployment Boundary

A.2.8.5 Components Interfacing with Equipment

A.2.8.5.1 Examples of Resource Layer Components

The Sensors, Effectors and Communicator resource components reason about the lowest level of specific equipment and so within a deployment their services may be utilised by other resource components that reason about other equipment resources or physical commodities. Examples of Resource Layer components are shown in Figure 60: Resource Layer Example.

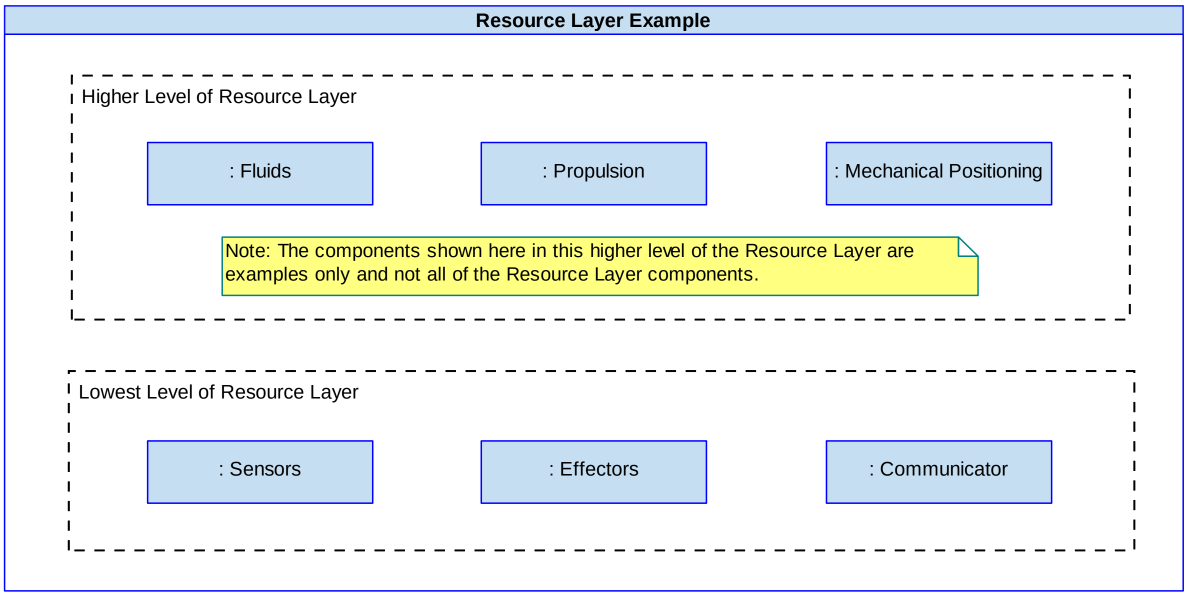

Resource Layer Example
Higher Level of Resource Layer
: Fluids
: Propulsion
: Mechanical Positioning
Note: The components shown here in this higher level of the Resource Layer are examples only and not all of the Resource Layer components.
Lowest Level of Resource Layer
: Sensors
: Effectors
: Communicator

Figure 60: Resource Layer Example

A.2.8.5.2 Sensors, Effectors and Communicator Components

The Sensors and Effectors components are used where there is a need for direct control or monitoring of equipment. These components interact with equipment of varying degrees of complexity, e.g. a solenoid valve (effector), a jamming pod (effector) or an Electro-Optic camera (sensor). The Communicator component too is used for direct control when there is a need to use communication equipment (e.g. a transceiver). Complex equipment may provide a relatively simple interface as the component does not need to fully understand its internal complexity (which may drive capability).

These three components may provide:

- High level state commands (e.g. power-on/standby/off).
- A translation of abstract commands into specific commands that can be understood by the equipment, based on the specific equipment interface definition (e.g. combining a requirement for a sensor to observe ‘target A’ with the information defining the location of ‘target A’, provided by another component, to form an instruction to observe a specific location).
- An interface conversion capability, typically where a non-digital interface interacts with equipment (e.g. defining a required signal voltage level that corresponds to a sensor angular positioning demand).
- A resource management capability, ensuring that demands for the use of the equipment do not conflict.

More complex items of equipment (but which still require direct control) might be considered as multiple instances of simpler equipment (e.g. individual actuators rather than ‘a flight control surface with its various actuators').

An example of this is shown in Figure 61: Vehicle Primary Flight Control Build Example.

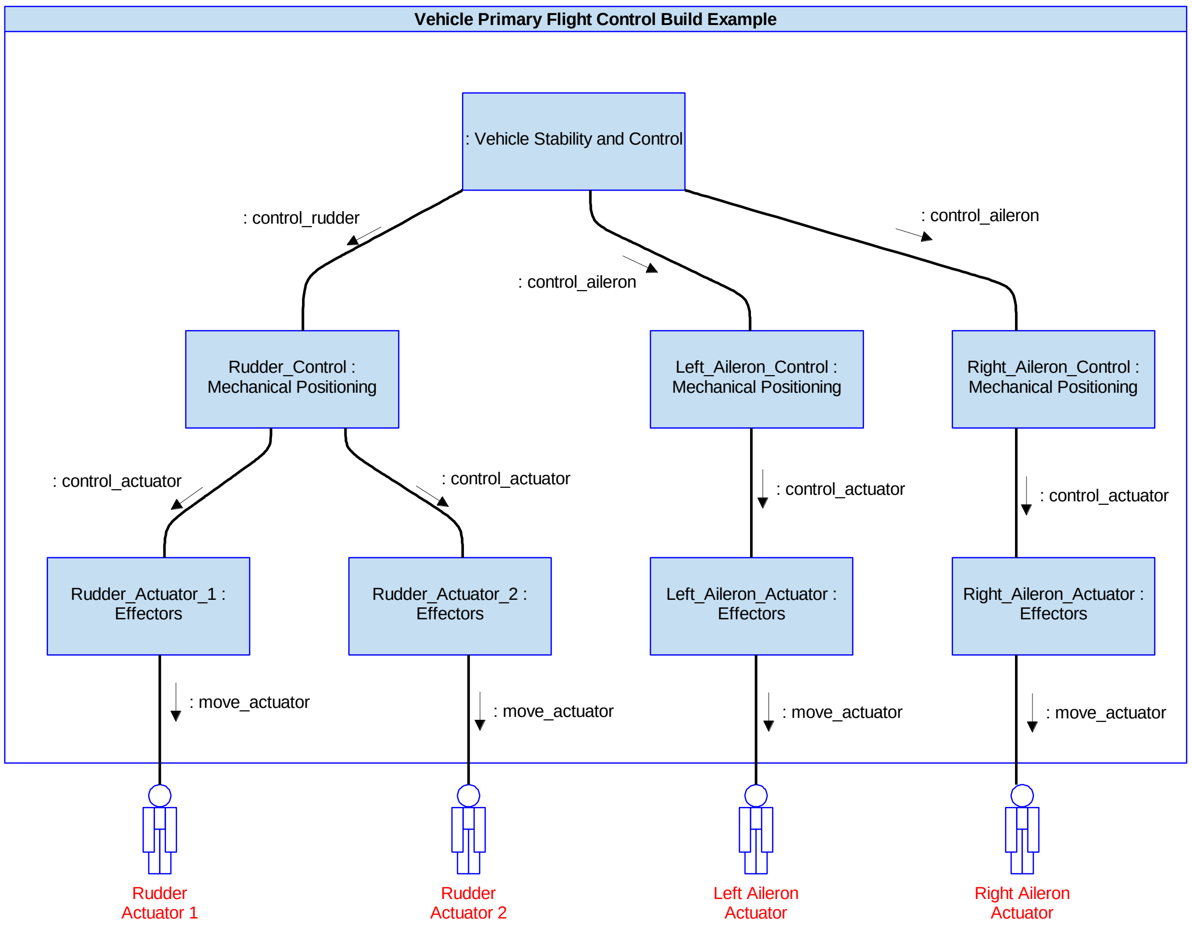

Vehicle Primary Flight Control Build Example
: Vehicle Stability and Control
: control_rudder
: control_aileron
: control_aileron
Rudder_Control : Mechanical Positioning
Left_Aileron_Control : Mechanical Positioning
Right_Aileron_Control : Mechanical Positioning
: control_actuator
: control_actuator
: control_actuator
: control_actuator
Rudder_Actuator_1 : Effectors
Rudder_Actuator_2 : Effectors
Left_Aileron_Actuator : Effectors
Right_Aileron_Actuator : Effectors
: move_actuator
: move_actuator
: move_actuator
: move_actuator
Rudder Actuator 1
Rudder Actuator 2
Left Aileron Actuator
Right Aileron Actuator

Figure 61: Vehicle Primary Flight Control Build Example

Figure 61: Vehicle Primary Flight Control Build Example shows how the Mechanical Positioning and Effectors components can be used in the primary flight control of an air vehicle. In this case, because the ailerons and rudder are specific to the system and are unlikely to change, the Mechanical Positioning component has been data-driven to create three component instances, two that represent the left and right ailerons which will be controlled in the same way and a third to represent the rudder that can be controlled differently and provide different capabilities. The Effectors component has been data-driven to create four component instances, one for each of the physical actuators.

A.2.8.6 Equipment

A.2.8.6.1 Simple and Complex Equipment

From the perspective of the PRA, the complexity of a piece of external equipment refers to the number of components which are required to provide the control interface.

Note that the level of internal complexity of the equipment is not considered by the PRA - for example, a sophisticated jammer might have a simple ‘on/off switch’ from the perspective of the PRA because all further decision making about how to employ the jammer occurs within the equipment. Despite its internal complexity, in the context of the PRA it remains a simple, Resource Layer controlled equipment, as shown in Figure 62: RF Jammer with a Simple Interface Example.

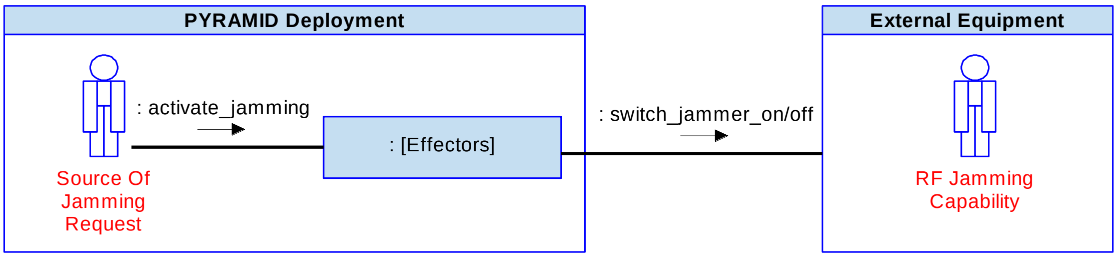


Figure 62: RF Jammer with a Simple Interface Example

A.2.8.6.2 Smart Equipment and the PYRAMID Deployment-Equipment System Boundary

The concept of 'smart' equipment refers to equipment that, rather than being exclusively controlled by the Sensors, Effectors, and Communicator components, can accept requirements from other components, including those at a higher layer in the Control Architecture (e.g. Task or Action Layer requirements).

The components that interface with the equipment may vary depending on the complexity and capability which resides within a specific piece of equipment. 'Smart' equipment is able to interact with higher layers of the Control Architecture because it has a degree of intelligence that enables it to interpret abstract requests corresponding to those layers, and is also able to control some, or all, of its own hardware without needing to be instructed by resource component(s).

Note that 'smart' equipment that receives requirements at an Objective, Task or Action Layer may still also interface with the components in the Resource Layer - e.g. to request power.

Figure 63: The PYRAMID Deployment/Equipment Boundary shows three examples of equivalent pieces of equipment - in all cases the overall requirement and the hardware that will ultimately be used are the same, but in each case additional capability resides within the equipment. This means that the PYRAMID deployment boundary is drawn at a different point and hence the requirements and information provided to the external equipment by the PYRAMID deployment varies.

Note that the components are shown inside the equipment for illustrative purposes only. The equipment need not contain PYRAMID compliant components, but will contain equivalent functionality.

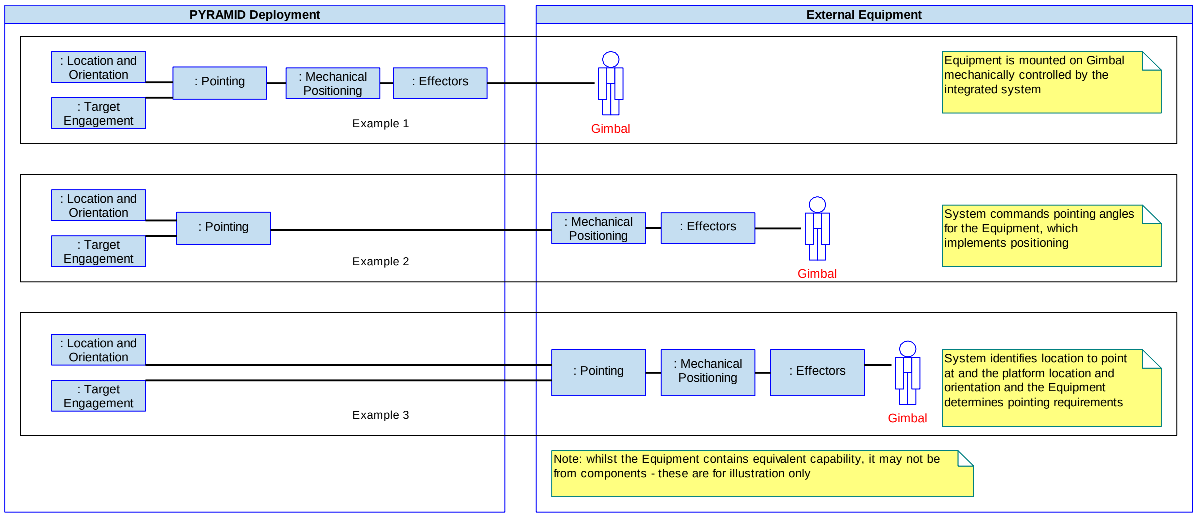

PYRAMID Deployment
External Equipment
: Location and Orientation
: Target Engagement
: Pointing
: Mechanical Positioning
: Effectors
Example 1
Gimbal
Equipment is mounted on Gimbal mechanically controlled by the integrated system
: Location and Orientation
: Target Engagement
: Pointing
Example 2
: Mechanical Positioning
: Effectors
Gimbal
System commands pointing angles for the Equipment, which implements positioning
: Location and Orientation
: Target Engagement
Example 3
: Pointing
: Mechanical Positioning
: Effectors
Gimbal
System identifies location to point at and the platform location and orientation and the Equipment determines pointing requirements
Note: whilst the Equipment contains equivalent capability, it may not be from components - these are for illustration only

Figure 63: The PYRAMID Deployment/Equipment Boundary

A.2.8.6.2.1 Integrating with Equipment

Each type of hardware item which a particular component will interface with has some commonality in the capabilities it provides and the way it is controlled. For example, a Paveway IV bomb and an AMRAAM missile will both require target and navigation information so a component that produces this information would have a standard interface to transfer this priming information.

When a piece of external equipment is integrated with a PYRAMID deployment, interfacing PRA components need to know its capabilities and the control commands required to deliver those capabilities to the relevant components within the PYRAMID deployment. This could be achieved through the data driving of a resource component to define the capability and control functions of a piece of hardware. Equipment that has been designed to be PYRAMID compliant may have the ability to publish its capabilities directly. For other equipment, e.g. legacy equipment, this type of functionality may not fully exist, if at all, and in that case the relevant interfacing component, e.g. Sensing, will use data driving to capture and report the equipment capability. Further information on data driving can be found in the Data Driving PYRAMID concept.

Adding a new item of equipment that interfaces with the same components as an existing item of equipment will have no impact on other (non-interfacing) components in the PYRAMID deployment, as the equipment capability is reported to the same components that are responsible for that interface.

If a new piece of equipment provides capability to a component that was not previously involved in an equipment interface, then if that component is already present, there should be no changes required to other (non-interfacing) components in the PYRAMID deployment.

If a new piece of equipment provides capability not understood by the PYRAMID deployment, then the appropriate component needs to be added to the deployment to control the interface.

Note:

- The approach outlined in this PYRAMID concept is a general approach.
- PYRAMID components may not control all equipment or all the interfaces used by equipment. For example, it may be appropriate for the infrastructure to control some equipment interfaces (e.g. processor cooling fans).

A.2.8.6.3 Examples of Equipment

High-level examples of interfaces between a PYRAMID deployment and external equipment are shown in the following sub-sections. For a more detailed discussion of the types of interfaces which may exist with deployable external equipment, refer to the Interfacing with Deployable Assets PYRAMID concept.

A.2.8.6.3.1 SATCOM

The equipment in these examples is a UHF Transceiver on a gimbal assembly, which is assumed to have low and high power output modes for transmission controlled by a safety critical discrete (as high power can cause injury to people), and to require encryption and/or high integrity protection which is performed outside the SATCOM equipment.

In the first example (Figure 64: SATCOM Equipment (Direct Control Example)), the gimbal direction is given to the equipment, whilst in the second example (Figure 65: SATCOM Equipment (High Level Control Example)) the equipment is given satellite location plus vehicle location and attitude from which it is able to calculate gimbal direction itself.

Note that in this example, by moving to a piece of equipment with greater internal capability, the number of components involved in the interface increases to six components rather than five, as the equipment requires the location of the platform and destination to determine pointing requirements rather than being provided with a pointing angle determined within the PYRAMID deployment.

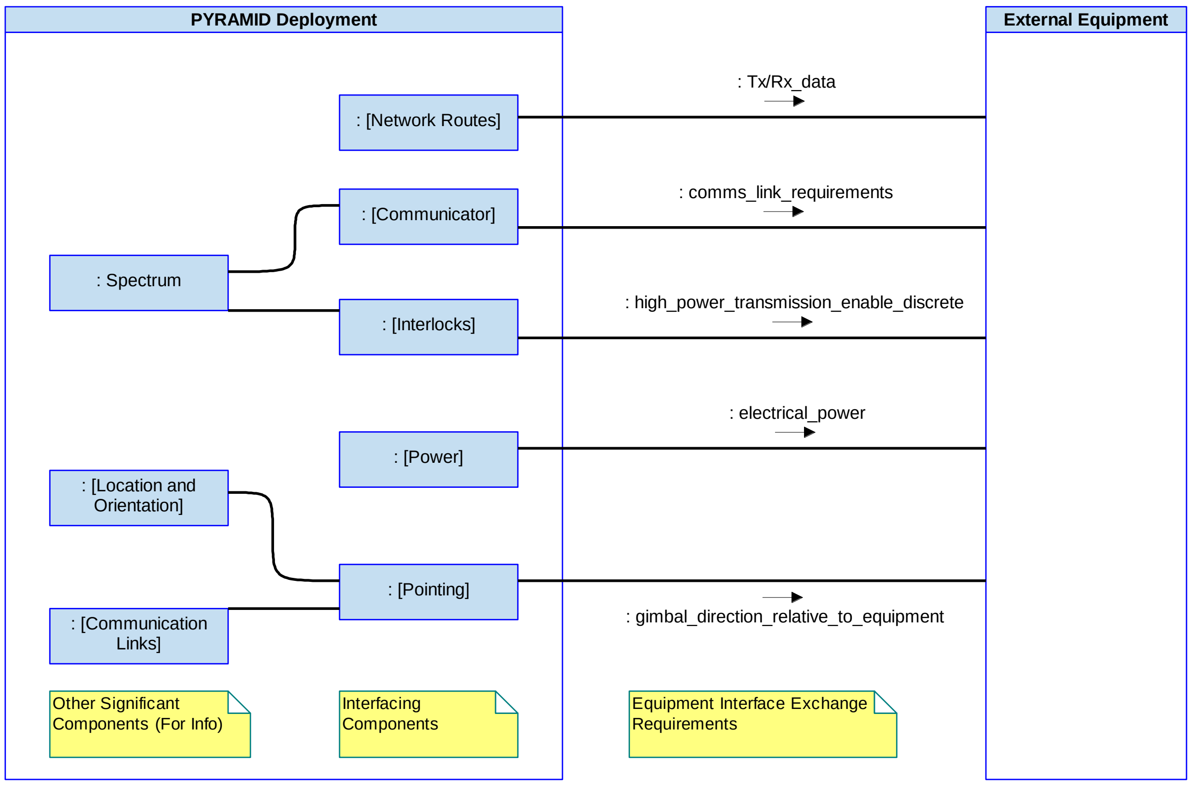

PYRAMID Deployment
External Equipment
: [Network Routes]
: [Communicator]
: Spectrum
: [Interlocks]
: [Power]
: [Location and Orientation]
: [Pointing]
: [Communication Links]
: Tx/Rx_data
: comms_link_requirements
: high_power_transmission_enable_discrete
: electrical_power
: gimbal_direction_relative_to_equipment
Other Significant Components (For Info)
Interfacing Components
Equipment Interface Exchange Requirements

Figure 64: SATCOM Equipment (Direct Control Example)

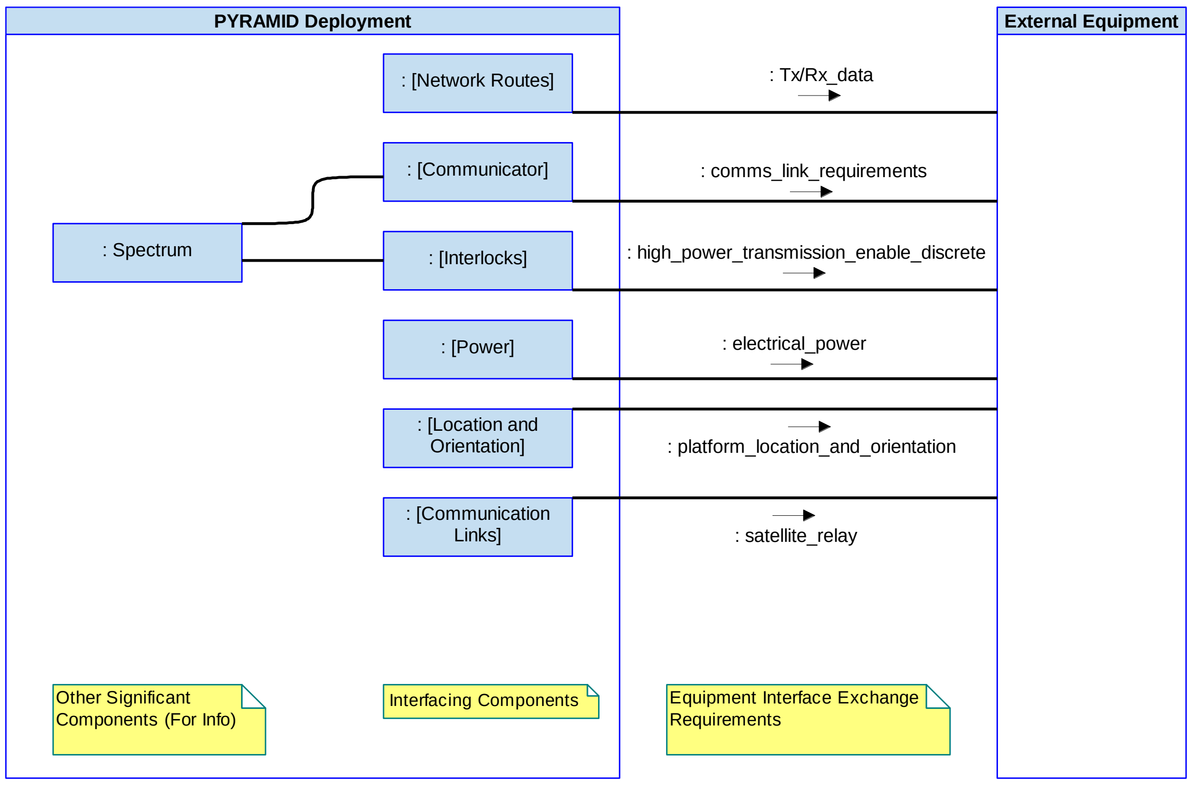

PYRAMID Deployment
External Equipment
: [Network Routes]
: [Communicator]
: Spectrum
: [Interlocks]
: [Power]
: [Location and Orientation]
: [Communication Links]
: Tx/Rx_data
: comms_link_requirements
: high_power_transmission_enable_discrete
: electrical_power
: platform_location_and_orientation
: satellite_relay
Other Significant Components (For Info)
Interfacing Components
Equipment Interface Exchange Requirements

Figure 65: SATCOM Equipment (High Level Control Example)

A.2.8.6.3.2 Laser Rangefinder and Designator

The equipment in these examples is a multifunctional laser designator pod. It incorporates an electro-optical sensor, laser rangefinder and laser designator, all mounted on a gimbal assembly to point it in the required direction. The equipment is assumed to have non-eye-safe lasers controlled by a safety critical discrete (as they can cause injury to people), and the capability to lase with a specific pulse repetition frequency code when commanded.

In the first example (Figure 66: LDP Equipment (Direct Control Example)), the required gimbal direction is provided relative to the equipment until a target is designated and electro-optical track mode is selected, and then the laser rangefinder is fired when commanded.

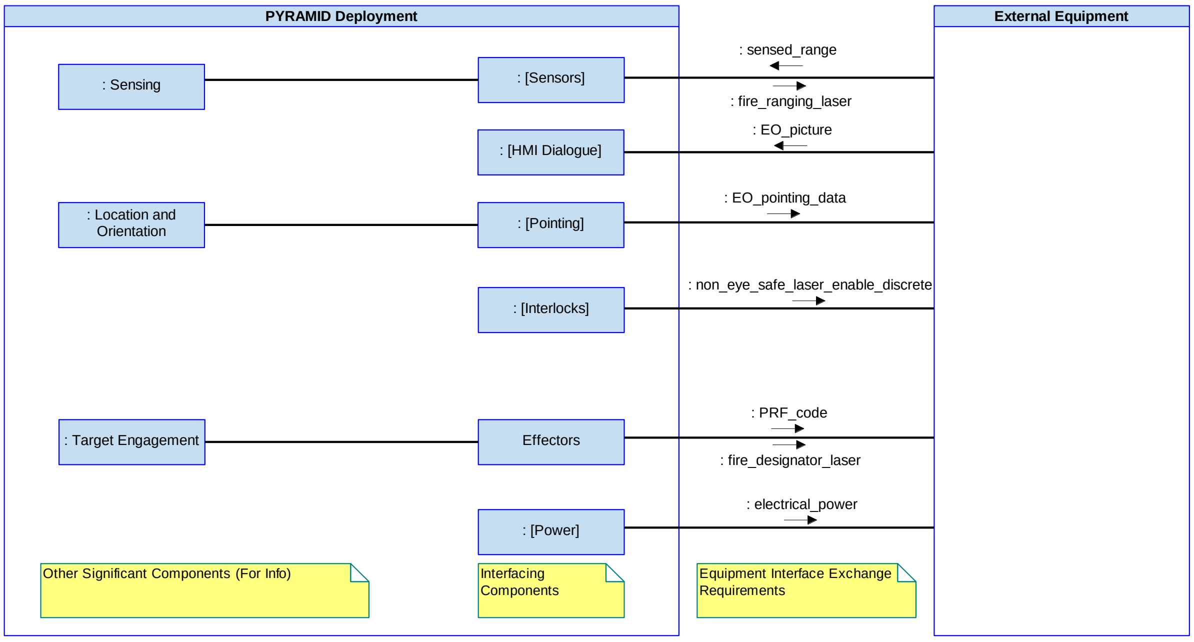


Figure 66: LDP Equipment (Direct Control Example)

In the second example (Figure 67: LDP Equipment (High Level Control Example)) the equipment is set to a search mode and provided with the area of interest and the characteristics of objects of interest (e.g. tanks), relying on the PYRAMID deployment to provide the location and orientation of the platform. The equipment determines whether it needs to use an IR mode, and negotiates with the PYRAMID deployment (as per any request for resource, as described in the Resource Management PYRAMID concept) to allow for the increased power demand of this mode. It returns tracks and track characteristics to the PYRAMID deployment in ‘real-world’ Earth coordinates. The PYRAMID deployment can direct it to focus on a particular track and lase it, but the LDP itself determines exactly when to fire its laser rangefinder. The HMI Dialogue and Effectors components have been left in this example to show how the equipment could also be controlled at a lower level, including directly by the operator.

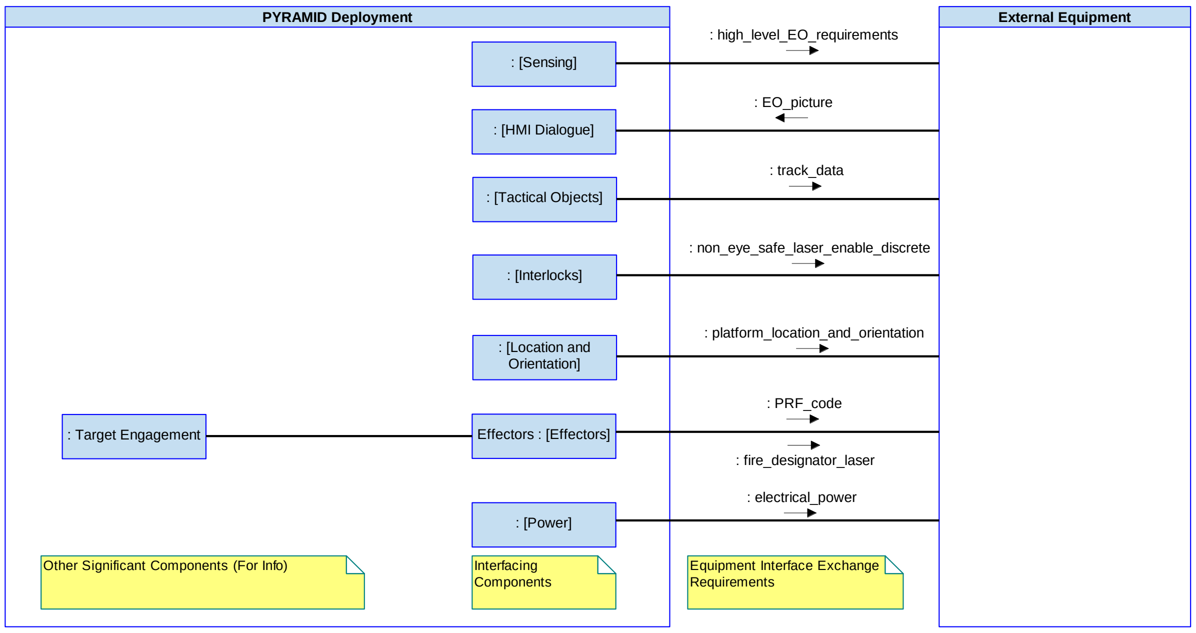


Figure 67: LDP Equipment (High Level Control Example)

A.2.8.7 Interaction with Equipment Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Interaction with Equipment Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept does not identify any generic component behaviours related directly to interaction with equipment. However, any component may support the interface to an external equipment by providing the services and functions that are required to support the interface and that are consistent with the defined subject matter of the PRA component. Consequently: • The PRA components used to interface to an external equipment should be selected according to whichever subject matter is appropriate given the specific nature of the equipment interface. • The PRA components used to interface with an equipment may therefore vary depending on the complexity and capability which resides within a specific piece of equipment. • Each aspect of an equipment interface should be supported by the appropriate PRA component, according to the relevant subject matter. • A PYRAMID deployment's interface with an equipment could therefore involve one or more PRA components. • Where one or more PRA components are used to support the interface to an external equipment, it is the responsibility of each component to understand the capabilities of the equipment, and the control commands associated with that capability, that fall within its subject matter.	In principle, any PRA component can interface to external equipment where the nature of the interaction is within the subject matter of the component. The PRA does not define any generic component responsibilities or services related directly to interaction with equipment. Where a component provides an interface to an external equipment, a corresponding service may or may not be defined by the PRA. Where the interface is not at the PRA scope boundary, and the function provided by the external equipment could have been provided by a PYRAMID component, then an associated service will be included in the PRA. Where the interface corresponds to the PRA scope boundary, services will not be defined by the PRA (see PYRAMID Technical Standard, Ref. [1], section Services Crossing the PRA Scope Boundary). The use of multiple components may therefore be required to support a single equipment interface, where the service interactions with the equipment are consistent with the subject matter of the respective components.
Specific PRA Component support for the	The PYRAMID concept does not identify any specialised subject matters in relation to interaction with equipment.	N/A.
Exploiter considerations	This PYRAMID concept does not identify any specific Exploiting Programme considerations.	N/A.

Table 9: Interaction with Equipment PYRAMID Concept Summary

A.2.9 Resource Management

A.2.9.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

- Control Architecture

Read in conjunction with

- Interaction with Equipment
- Dependency Management
- Constraint Management

A.2.9.2 Introduction and Scope

This PYRAMID concept explains how the principles of resource management can be applied through the use of the PRA, and applies to managed resources available to the system.

A.2.9.2.1 What is a Managed Resource?

In the context of the PRA, a managed resource is a resource whose availability to the system is limited and whose use by the system needs to be managed. The finite nature of a resource may be due to an aspect of the platform (including consumables and equipment) or, in special cases, something external to the platform that the platform is inherently dependent upon (such as the electromagnetic spectrum). There are two primary aspects to the finite nature of a managed resource that need to be considered:

- **Limited by total use over time:** Many resources will need to be managed to ensure their availability over time. Examples include fuel, stores and the energy stored in a battery. The use of such resources reduces availability for future use, though for some types of resources it may be possible to replenish supplies during operation. The management of resources that are limited over time requires accountancy of the consumption and replenishment of the resource over the course of operations.
- **Limited by instantaneous capacity for use:** Many resources will need to be managed to ensure that instantaneous demand for the resource can be met. Examples include access to a sensor, communicator or effector, or the instantaneous limits on the power delivered by an electrical power supply. Limitations on instantaneous use may be defined by:
  - Sharing rules or capacity limits such as simple limits on the number of users of a resource (e.g. use of a sensor).
  - A constraint on the type of simultaneous use (e.g. a multi-functional array that provides both a sensor and communicator capability but cannot act in both modes simultaneously).
  - The total instantaneous demands for the resource from one or more users (e.g. a power supply).

Some types of resource may also be scalable to enable control of instantaneous capacity, where this is adjustable or the resource is available from multiple sources. The management of resources with instantaneous limits of use necessarily requires scheduling and accountancy of instantaneous demand and the management of scalability controls.

Many resources will need to be managed in both these respects, for example, a battery has both an instantaneous power rating and its energy is depleted over time.

A.2.9.2.2 What is Resource Management?

Given the finite nature of managed resources, the goal of resource management is to try to optimise the allocation of managed resources, to meet demands, in a way that reflects the priorities of the system.

The limited availability of managed resources makes it desirable to:

- Minimise their use in certain cases (e.g. fuel usage).
- Restrict their use in cases (e.g. a steerable antenna is a resource that cannot be used by multiple users simultaneously).
- Maximise their benefit while in use (e.g. by considering the placement of a sonobuoy which once in use does not stop until fully exhausted or by prioritising the message transmissions going through a communications device).
- Identify the need for resource replenishment and to plan and manage replenishment.

Resource management is the process of capturing the resource needs of the system and matching them to resource availability so that an acceptable level of utility is achieved for a bearable cost. This process includes the following primary steps:

- **Resource Estimation:** Provide an estimate of the resource requirement, specifying what is needed and when it is needed.
- **Resource Allocation and Scheduling:** Determine a solution to satisfy resource requirements, defining the allocation of resources over time.
- **Identification of Resource Conflicts:** Identify conflicting resource requirements and characterise the nature of the conflict.
- **Resource Brokering:** Mediate a resolution to resource demand conflicts to maximise the utility of resource allocation.
- **Resource Arbitration:** Make prioritisation decisions where resource conflicts cannot be resolved by other means.
- **Determine Resource Availability:** Identify possible providers of a resource, their capability to provide that resource and any constraints on their use.
- **Resource Monitoring:** Monitor resource status, including determining the current and predicted resource availability for use, and monitoring usage against allocations and planned replenishment to enable re-planning (resulting from new or changed requirements, or predicted shortfalls).

- **Changing Resource Availability:** Taking steps to change resource availability, such as planning resource replenishment or altering the characteristics of a resource provider (e.g. if it has internal settings, which make it less capable but more efficient, for greater total resource availability over time).

A.2.9.2.3 Why is Resource Management Important?

When resources can be shared or used to perform multiple activities, actively identifying and resolving conflicts provides a flexible and efficient solution to resource usage, making it more likely that mission objectives can be met within an acceptable resource cost.

- The conservative allocation of managed resources could lead to a very predictable system, but one that is potentially wasteful and inflexible. Sometimes this is the best strategy for high integrity systems.
- Inflexible allocation can lead to a managed resource shortfall and thus an underperforming system, e.g. due to unexpected demand or loss of managed resource.
- Poor managed resource estimation could lead to undetected over or under allocation and unpredictable performance.
- Failure to correctly prioritise allocation can lead to an underperforming system.

A.2.9.3 Overview

The PYRAMID concept for Resource Management can be summarised as follows:

- Strategies for resource management have been developed based on the need to manage the instantaneous capacity of resources at a single point in time and the total use of the managed resource over time, as defined above.
- Components will implement these strategies by taking on one or more specific roles with defined standard behaviours.
- Components will provide traceability of resource demands they identify to the related requirement placed on them, allowing the origin of a resource demand to be identified, even if the originator is not aware its requirements result in a resource demand.
- The General Conflict Resolution Pattern (as defined in Figure 32: General Conflict Resolution Pattern, in the Dependency Management PYRAMID concept) is applied when resource conflicts cannot be resolved by the component responsible for allocating the particular type of resource.

A.2.9.4 Component Composition

This PYRAMID concept identifies a number of generic component behaviours in relation to Resource Management. These behaviours are formally captured as component responsibilities in the PYRAMID Technical Standard, section Component Composition, along with associated component services. These responsibilities and services are also summarised in Table 10: Resource Management PYRAMID Concept Summary.

Appendix B: Use Cases demonstrates how these responsibilities and services support component interaction in relation to Resource Management.

A.2.9.5 Resource Management Pattern

The General Conflict Resolution Pattern (as defined in Dependency Management PYRAMID concept, Figure 32: General Conflict Resolution Pattern) is directly applicable to the placing of demands for resource use and resolving conflicts between demands. Figure 68: Component Roles in Resource Management specialises this pattern specifically for resource management, by showing the generic roles that are relevant to the resource allocation process. Implementing traceability, from originating demands to the need for resource allocation, is an important aspect of resolving resource demand conflicts.

A.2.9.5.1 Resource Management Roles

The following generalised component roles are used in this PYRAMID concept:

- **Indirect Resource User:** Generates requirements that indirectly generate resource demands.
- **Direct Resource User:** Directly generates requirements for the use of a resource or a resource capability that result in resource allocation demands.
- **Resource Provider:** Determines solutions involving resource use, including determining and allocating resource demands, seeking to optimise resource allocation and identifying demand conflicts.
- **Resource Broker (the Conflict Resolution Component):** Acts as an intermediary to facilitate resolving resource demand conflicts and perform or request arbitration (see section Resource Management and the Control Architecture PYRAMID Concept for a summary of the arbitration process).

Individual components may operate in more than one of the above roles. For example, a Direct Resource User may also be an Indirect Resource User, generating dependent requirements. A Resource Provider may also be a Direct Resource User, generating demands for a different type of managed resource. Figure 68: Component Roles in Resource Management shows how components may take on different resource management roles when placing or satisfying requirements. Where components sit within this structure affects their level of understanding of the resources being managed. For example, an Indirect Resource User may place requirements on to an action component with no knowledge of any consequential resource demands. The resource management strategies described in the Strategies for Managing Resources section explain how components, at all levels, can contribute to resource management in their own terms.

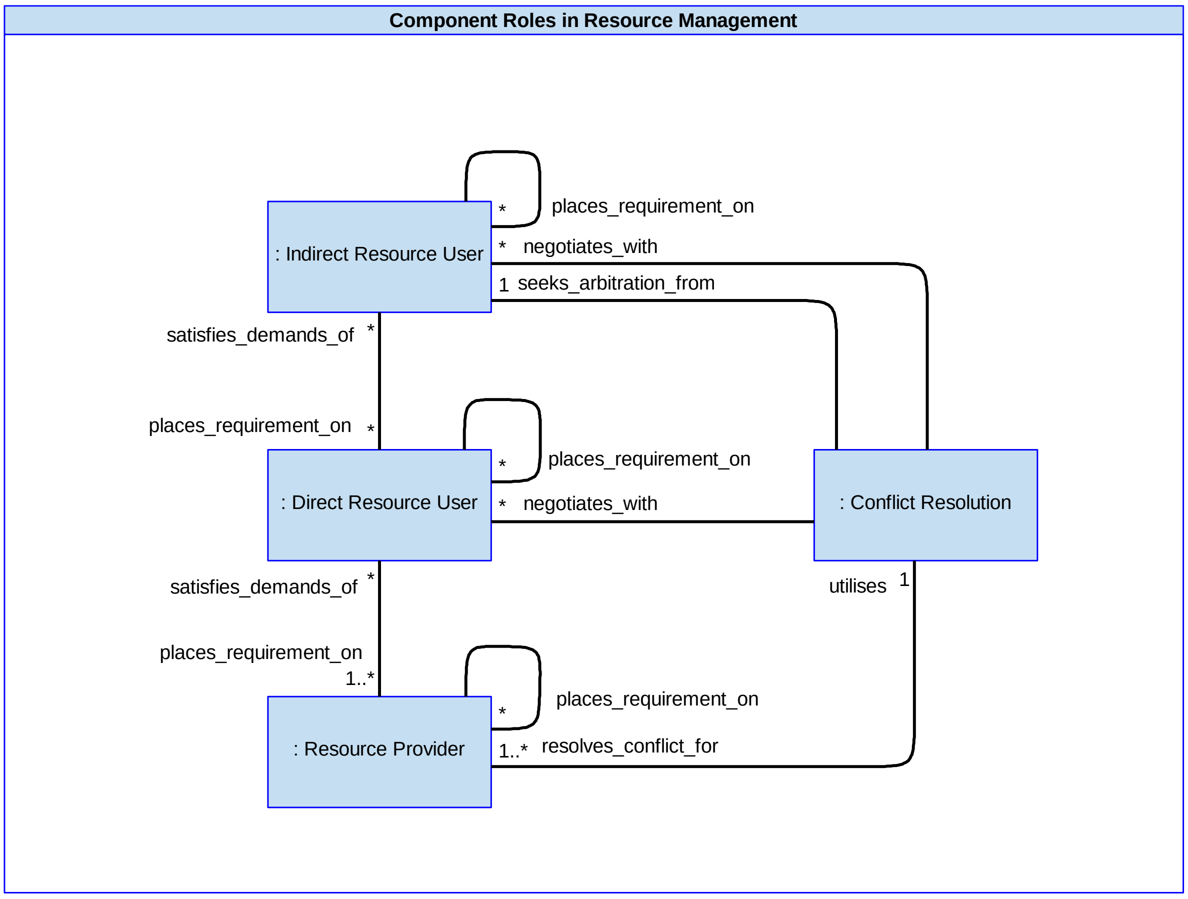


Figure 68: Component Roles in Resource Management

A.2.9.5.2 Demand Traceability

Objectives are decomposed through the different layers of the control architecture, including the Task, Action and Resource Layers, see the Control Architecture PYRAMID concept. The plan to meet a given objective, will typically rely on multiple tasks and associated actions and the use of resources being planned across the system. Each individual resource demand needs to be traceable back to the requirement that gave rise to it and the particular proposed solution it forms part of (see Dependency Management PYRAMID concept, section Deriving Solutions), such that all resource allocations associated with the same overall requirement can be identified. The components involved in resource management require an understanding of resource demand traceability as follows:

- **Indirect Resource Users and Direct Resource Users:** Maintain traceability of requirements placed on them and their requirement dependencies.
- **Resource Provider:** The determination of resource allocation solutions, and correspondingly the identification of demand conflicts, requires that Resource Providers can identify and collate related requirement sets. This requires an understanding of requirement traceability to a common overarching activity. Equally, alternative solutions may be developed within the system which are mutually exclusive and for which the corresponding resource demands cannot, by definition, be in conflict. The process of resource monitoring also requires that the Resource Provider maintain a mapping of resource allocations and usage to their requirements.

- **Conflict Resolution Component:** In order for the Conflict Resolution component to facilitate the resolution of conflicts, whether through re-planning or arbitration at any layer of the General Conflict Resolution Pattern, it must determine the traceability of resource requirements to their point of origin, including the traceability of resource dependencies.

A.2.9.6 Strategies for Managing Resources

A.2.9.6.1 General Strategies

The general approach for resource management is a specialisation of the General Conflict Resolution Pattern (see the Dependency Management PYRAMID concept), facilitated by the Conflict Resolution component. Mutually conflicting resource demands (as well as conflicts between resource demands and constraints) are identified by the affected resource provider and Conflict Resolution proceeds through a process of brokering and arbitration as with any other conflicting demand on a component.

A.2.9.6.2 Resources with Limited Total Use Over Time

The management of a resource's total use over time consists of maintaining a resource budget by monitoring and controlling the total available quantity of a resource and tracking its usage and replenishment. The General Conflict Resolution Pattern (see the Dependency Management PYRAMID concept) supports this by resolving conflicting demands where the Resource Provider cannot accommodate all demands within its managed resource budget.

Replenishment: Certain managed resources are also capable of being replenished during operation and the management of such resources requires that the need for replenishment can be identified and therefore planned for. In some situations, carrying out a replenishment activity might mean that current or planned mission requirements are impacted or detracted from, or in order to replenish the resource a different resource is consumed. Resource components should be made aware of any plans that result in their resource being replenished, so that they can take account of this increased availability when allocating their resource. Likewise, where resource demands cannot be satisfied, due to a lack of availability, resource components can indicate the resource shortfall, so that other components, such as Tasks, can take actions to replenish the resource where necessary. For example, during mission planning the calculated fuel demands may exceed the vehicle capacity and solution refinements through brokering may fail to resolve the conflict. Arbitration in this case may not be an option since this would undermine the goals of the mission with all elements of the plan considered essential. This leads to the identification of the need for resource replenishment, and thus the planning of a refuelling activity.

A.2.9.6.3 Resources with Limitations on Instantaneous Use

The management of resources with limitations on instantaneous use (as defined in section What is a Managed Resource?) follows the same general pattern defined in the General Conflict Resolution Pattern (see the Dependency Management PYRAMID concept) and are subject to the following considerations:

- Scheduling to avoid conflict and conform to limits on simultaneous use at the same point in time.

- Scheduling to account for the time to transition between resource users.
- Control of capacity where this is adjustable or the managed resource is available from multiple sources.

Such managed resources are subject to time scheduling to avoid conflicting demands whilst also conforming to any applicable sharing rules. The sharing rules for each managed resource are owned and applied by the Resource Provider, and must define the limitations on the instantaneous use of the managed resource. The limitations on instantaneous use may constrain the number or combinations of simultaneous users, or place some other limit on instantaneous capacity.

Examples of different limitations on simultaneous use include:

- A sensor array which can only be allocated to a single user at once.
- A power supply which can support any number of users provided the total power consumption never exceeds an instantaneous capacity limit.

There may also be more complex limitations on simultaneous use of a resource where particular combinations of simultaneous use are restricted, such as a spectrum where frequency bands can be used by different users at once provided rules on mutual interference are followed. In addition, there are considerations that may affect allocations beyond the demands placed on it by the Resource Users:

- **Transition Times:** Certain resources may be subject to finite transition times to switch between users, for example to allow the reconfiguration of equipment. The understanding of these transition periods, and allowance for them when scheduling resource use, is the responsibility of the Resource Provider.
- **Resource Scalability:** A resource may have adjustable capacity limits through the control of a single source or the availability of multiple sources. This could enable instantaneous capacity to be increased to satisfy demand, potentially at the expenses of other managed resources or the availability and capacity of supply at some future time. For example, the amount of electrical power generated could be increased at the expense of additional fuel use, or the amount of electrical power made available could be increased through the use of a reserve battery. The understanding and control of managed resource supplies in this way is also the responsibility of the Resource Provider.

For resources that have limitations on instantaneous use, a Direct Resource User places its requirement on to the Resource Provider, specifying both what is needed and when it is needed (including any flexibility in scheduling, such as specifying a required period within a given time window). The Resource Provider attempts to determine a scheduling solution that satisfies all the requirements placed on it, while applying the appropriate sharing rules and managing the other considerations described above. The Resource Provider may find a solution to satisfy requests, creating a non-competing schedule within the flexibility afforded by individual demands. However, where the Resource Provider identifies a conflict, Conflict Resolution is invoked to facilitate a resolution. This may involve a negotiated local resolution in which Direct Resource Users are able to re-plan locally to change the resource demand. Here the solution refinement goals may target schedule adjustments, reductions in instantaneous demand or some other change in the managed resource demand, for example, modifying the sensor capability required enabling an alternative, and available, sensor to be used.

Where Direct Resource Users are unable to re-plan, higher level conflict resolution may be sought. For example, the Indirect Resource User may be asked to re-plan to meet revised scheduling goals. In the absence of a successful resolution, a process of arbitration may be undertaken.

It is possible that a planned allocation solution might have to deal with an interruption where a demand cannot be scheduled in advance (such as the need to use a frequency allocation for defensive jamming):

- **Planned Interruption:** If the specific timing of a managed resource demand cannot be fixed during planning, it would be very inefficient to prevent other components from using the resource over the period where its availability is indeterminate. A solution to this is to allow other components access to the managed resource with the proviso that access may be interrupted. Note that this strategy is intended to address potential interruptions due to demands which are known about in advance but whose timing cannot be planned precisely. Such an arrangement needs to recognise the priorities of the demands and the consequences of the interruption. That any given resource requirement includes the flexibility to be interrupted in this way (or otherwise) can form part of the initial resource requirement specification or be subsequently agreed as part of a conflict resolution process negotiated by Conflict Resolution. Contingency planning is supported by this mechanism of planned interruption, enabling a managed resource to be allocated for use with the understanding that it may be reallocated with immediacy to support a contingency measure.
- **Unplanned Interruption:** Unplanned interruptions caused by, for example, a loss of system capability may fall under other PYRAMID concepts such as Capability Management and Dependency Management. However, an unplanned interruption due to a new urgent requirement would be handled via Conflict Resolution as described above, i.e. the Resource Provider may recognise that the new requirement conflicted with a current allocation and Conflict Resolution may facilitate a resolution. In the case of an urgent high priority demand (or other time-critical decision where the normal planning process cannot be followed), this might entail an immediate arbitration decision depending on the rules and strategies configured within Conflict Resolution (see example in section Conflict Arbitration (Unplanned Interruption)).

A.2.9.7 Resource Management and the Control Architecture PYRAMID Concept

The Control Architecture PYRAMID concept illustrates how components in the Action Layer make use of components in the Resource Layer to carry out their actions. This behaviour could be carried out without active resource management, but this would not always maximise the use of available resources or ensure optimal functional capability.

The different layers of the Control Architecture may be involved in resource management as follows:

- **Task Layer:** The Tasks component will typically be the originator of many action requirements that result in resource demands, thus acting in the role of Indirect Resource User. This may include contingency related activities that result in prioritised resource allocation/reservation. In support of resource conflict resolution, the Tasks component may be required to: modify one or more of the action requirements which lead to a conflict, impose constraints which indirectly limit the resource that can be used to achieve the action requirements, or support arbitration by making prioritisation decisions.

- **Action Layer:** Components in the Action Layer generate proposed solutions to satisfy the action requirements placed on them. These solutions may result in direct resource demands but may also have dependencies that result in further actions. Components in the Action Layer may therefore act as Direct Resource Users and Indirect Resource Users. In both cases, Action Layer components support conflict resolution by determining alternative solutions to meet refinement goals, and thus modifying the dependency requirements that led to a resource conflict, or by making prioritisation decisions. In addition, they may be responsible for determining the solution refinement goals for higher layers when resolving conflicts.
- **Resource Layer:** The Resource Layer contains components that determine solutions to satisfy the resource requirements placed on them. These solutions may also have dependencies that result in demands for other resources. Components in the Resource Layer may therefore act as Resource Providers and Direct Resource Users. Resource components reason either about equipment resources (at different levels of abstraction) or about non-equipment resources (such as fluids) and necessarily understand the availability of resources over time. In seeking solutions to satisfy requirements, resource components may identify conflicts and characterise the nature of conflicts. This characterisation supports high layer optimisation. In the first instance, a resource component will look to resolve such a conflict within any flexibility provided within the requirements placed on it. Where a Resource Provider identifies a conflict that it cannot resolve, the Conflict Resolution component is utilised*.

*Note that components will use a version of the Broker_Conflict service to request conflict resolution through brokerage and arbitration. See the service definition in the component composition (See PYRAMID Technical Standard, Ref. [1], section Component Composition).

A.2.9.8 Resource Management Examples

A.2.9.8.1 Limit on Total Use Over Time

In this example, Figure 69: Resource Management Example 1, there is a need to plan a new route within the limits of a remaining fuel resource. A significant portion of the fuel budget has already been allocated. Two possible scenarios are shown; one for conflict resolution by relaxation of requirements and one for conflict resolution by task prioritisation. The event and association highlighted in red is an optional step which is not part of the General Conflict Resolution Pattern. This step could be used in specific circumstances, this is explained further in the Conflict Brokering (Requirement Relaxation) example.

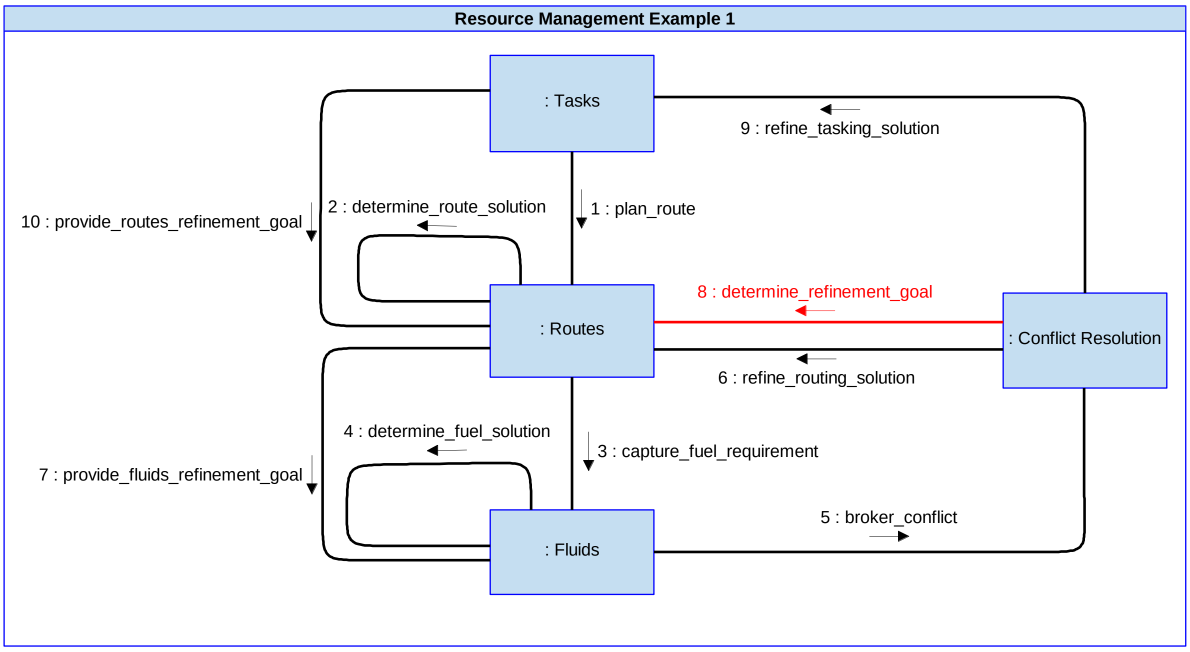

Resource Management Example 1
: Tasks
9 : refine_tasking_solution
10 : provide_routes_refinement_goal
2 : determine_route_solution
1 : plan_route
8 : determine_refinement_goal
: Routes
: Conflict Resolution
6 : refine_routing_solution
4 : determine_fuel_solution
3 : capture_fuel_requirement
7 : provide_fluids_refinement_goal
5 : broker_conflict
: Fluids

Figure 69: Resource Management Example 1

A.2.9.8.1.1 Conflict Brokering (Requirement Relaxation)

During a mission, a new requirement emerges to detour from the current route via a defined location. The Tasks component starts to plan the task, requesting that the Routes component finds a solution for the fastest route that accommodates the new waypoint within the defined constraints. Routes collaborates with other components (not shown) to find a solution and estimates the fuel required for it. The Fluids component captures this fuel requirement and determines that the quantity requested, in conjunction with other demands, exceeds the budget. Fluids therefore notifies the Conflict Resolution component and provides details of the requirements in conflict. The Conflict Resolution component determines that nothing within its configured rules warrants immediate arbitration and determines the brokering process that should be followed.

In accordance with the defined brokering process, the Conflict Resolution component triggers Routes to seek to refine its solution. Routes consults Fluids to determine the refinement goal and Fluids confirms the target reduction in the fuel demand that is required. Routes is unable to find an alternative solution to meet the refinement goal within the bounds of the existing requirements and constraints, and notifies the Conflict Resolution component. The Conflict Resolution component undertakes a similar exchange with Tasks triggering it to try and refine its solution. Tasks consults Routes to determine the refinement goal and Routes confirms the need to relax the fastest solution goal. Tasks determines an alternative solution to its own requirement that removes the fastest route requirement. Tasks then responds to the Conflict Resolution component that an alternative solution has been found and issues the revised requirement to Routes. Routes determines a solution to meet the revised requirement that results in a reduced fuel demand on Fluids. Fluids determines that the revised demand can be accommodated within the budget and notifies Routes that its resource dependency has been satisfied and notifies the Conflict Resolution component that the conflict has been resolved. The Routes component confirms to the Tasks component that a solution has been found and the plan is confirmed.

The event and association highlighted in red (step 8) is an optional step and not part of the General Conflict Resolution Pattern. There may be scenarios where the conflict is caused by a constraint being imposed on another component, the Conflict Resolution component is able to identify what component is constraining it and ask for the constraint to be removed or relaxed. For example, the Routes component might be given a constraint to avoid a particular region which impacts how it generates the route. By asking Routes for a refinement goal that would resolve the conflict Conflict Resolution is able to either request; Tasks to change the requirement it is placing on Routes, or it could ask the component constraining Routes to relax or remove that constraint.

A.2.9.8.1.2 Conflict Arbitration (Task Prioritisation)

During a mission, it becomes necessary to abort the current objective and to return to a specified location in the shortest possible time. The Tasks component re-plans, generating a high priority requirement for Routes to find the fastest route to the required location within the remaining fuel budget. Routes determines from Fluids the remaining fuel (regardless of current allocations) and collaborates with other components to find the fastest route solution within the specified budget. The Fluids component captures the resulting fuel requirement and identifies a conflict with current allocations, which it notifies to Conflict Resolution. Given the current mission context, the Conflict Resolution component's rules specify immediate arbitration, identifying the Tasks component as the arbiter. Tasks confirms to Conflict Resolution that the new requirement has sole priority and this arbitration decision is communicated to Fluids. Fluids confirms to Routes that the required quantity has been allocated. The Routes component confirms to the Tasks component that a solution has been found and the plan is confirmed. Tasks also rescinds the previous actions that have been superseded as a result of the mission abort.

A.2.9.8.2 Limit on Instantaneous Use

In this example, there is a need to plan a search task to detect and identify tactical objects over a large area. The search involves image capture and RF sensing over a broad frequency range. The use of communications is required for a significant portion of the mission. This type of task relies on managing the instantaneous use of RF spectrum which is used for both RF sensing and communications. The primary events are illustrated in Figure 70: Resource Management Example 2. Three possible scenarios are shown; resolving the conflict by schedule refinement, or having the conflicting requirement be a planned or unplanned interruption. Note that there are a number of things that need to be considered within this example:

- The interactions and events shown in red are only relevant to the third scenario (unplanned interruption).
- These scenarios are deliberately simplified for clarity and ignore many of the details and considerations that would need to be considered for sensing, communications and employing countermeasures. The scenarios are not intended to be a realistic representation of real mission considerations.
- These scenarios deliberately depict multiple ways of interacting with the Spectrum component to coordinate resources, through either Action layer components (e.g. Communication Links and Countermeasures) or Resource layer components (i.e. Sensors). An Exploiter would

need to choose the most appropriate way of interacting with Spectrum that will enable their system to coordinate resources in the most effective manner.

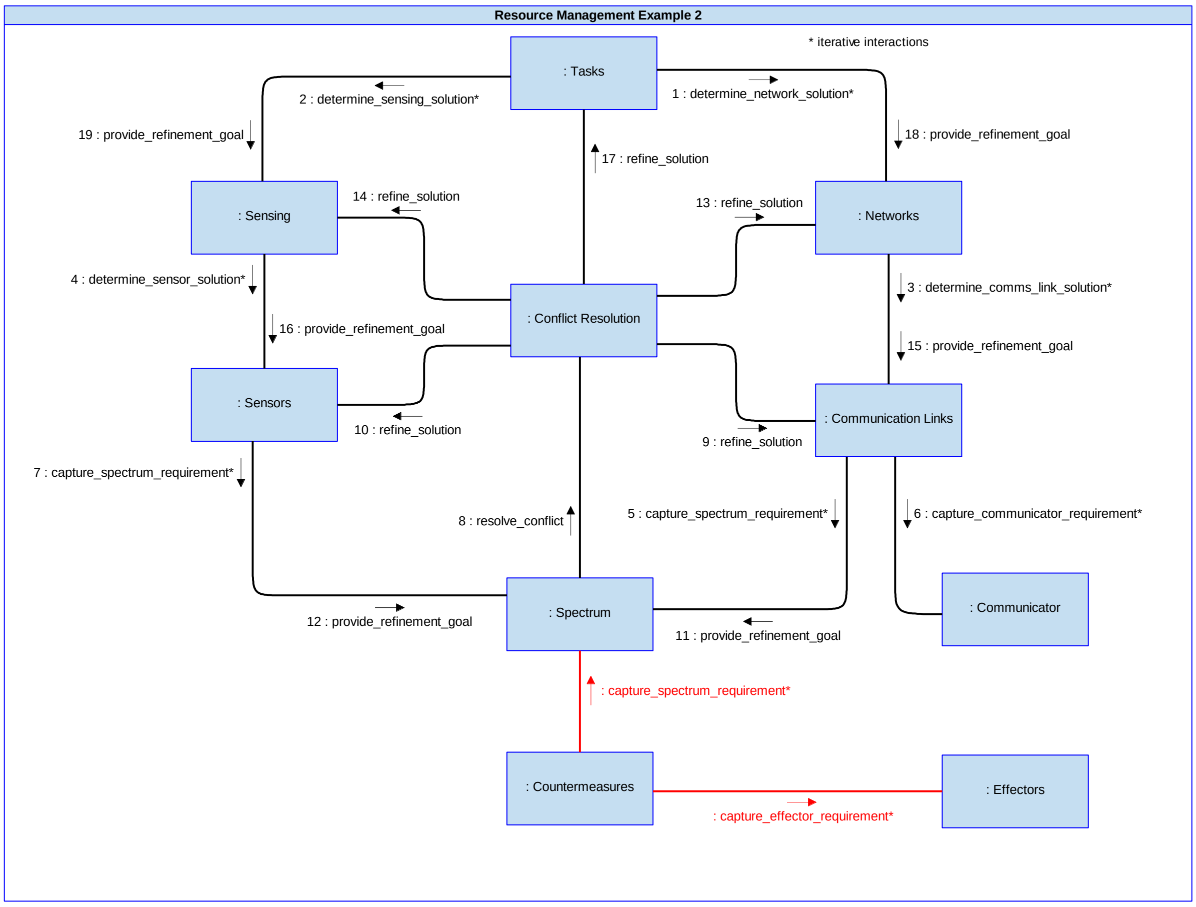

Resource Management Example 2
* iterative interactions
: Tasks
: Sensing
: Networks
: Conflict Resolution
: Sensors
: Communication Links
: Spectrum
: Communicator
: Countermeasures
: Effectors
1 : determine_network_solution*
2 : determine_sensing_solution*
3 : determine_comms_link_solution*
4 : determine_sensor_solution*
5 : capture_spectrum_requirement*
6 : capture_communicator_requirement*
7 : capture_spectrum_requirement*
8 : resolve_conflict
9 : refine_solution
10 : refine_solution
11 : provide_refinement_goal
12 : provide_refinement_goal
13 : refine_solution
14 : refine_solution
15 : provide_refinement_goal
16 : provide_refinement_goal
17 : refine_solution
18 : provide_refinement_goal
19 : provide_refinement_goal
: capture_spectrum_requirement*
: capture_effector_requirement*

Figure 70: Resource Management Example 2

A.2.9.8.2.1 Conflict Brokering (Schedule Refinement)

In this example, Tasks determines the need to plan both the sensing activity and the required communications and places requirements on to the corresponding Action components. Networks determines a network solution deriving requirements for supporting communication links. Communication Links determines a solution to meet this requirement deriving in the process the need for a specific Communicator capability and the use of a portion of spectrum. Spectrum confirms that the required spectrum can be allocated and Communication Links places its requirement on to Communicator. Communicator also accept the requirement and confirms to Communication Links that the communicator resources have been successfully allocated for the required time window. Communication Links confirms that it has successfully determined a solution to Networks and Networks correspondingly reports to Tasks.

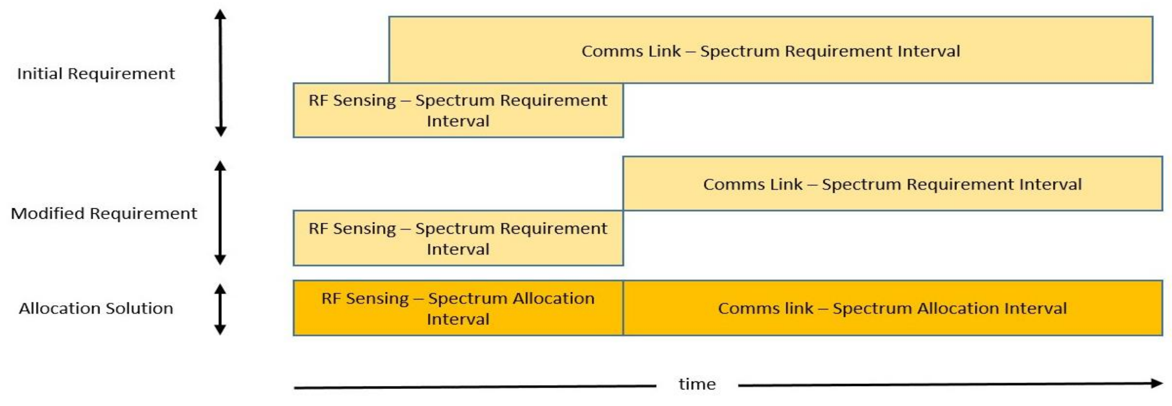

Initial Requirement
Comms Link – Spectrum Requirement Interval
RF Sensing – Spectrum Requirement Interval
Modified Requirement
Comms Link – Spectrum Requirement Interval
RF Sensing – Spectrum Requirement Interval
Allocation Solution
RF Sensing – Spectrum Allocation Interval
Comms link – Spectrum Allocation Interval
time

Figure 71: Scheduling Refinement

Sensing also determines a solution to meet its requirement, deriving in the process the need for particular sensor capabilities to support the image capture and RF sensing. Sensors accepts these sensor requirements and successfully determines a solution to undertake the sensor data capture. To enable the RF sensing solution, Sensors requests allocation of the required spectrum. The Spectrum component determines that the requested time of use conflicts with the existing (communications) allocation and that neither requirement provides any flexibility that might enable the conflict to be resolved locally. Spectrum therefore notifies the Conflict Resolution component and provides the details of the requirements in conflict.

The Conflict Resolution component determines that nothing within its configured rules warrants immediate arbitration and determines the brokering process that should be followed. In accordance with the defined brokering process, the Conflict Resolution component triggers both Sensors and Communication Links to seek to refine the relevant solutions. Each component consults Spectrum to determine the refinement goal and Spectrum confirms the nature of the scheduling conflict (both in time and frequency range). Sensors and Communication Links are unable to find an alternative solution to meet the refinement goal within the bounds of their existing requirements and constraints, and notify the Conflict Resolution component. Conflict Resolution undertakes similar exchanges with Sensing and Networks, which results in the identified need for corresponding relaxations of their timing requirement.

The Conflict Resolution component then triggers Tasks to seek to refine its solution and Tasks consults both Sensing and Networks to determine the refinement goal. Tasks determines an alternative solution that removes the time overlap between the actions, delaying the use of communications until after completion of the RF sensing activity. Tasks notifies the Conflict Resolution component that an alternative solution has been found and issues the revised requirements. Sensing and Networks, and subsequently Sensors and Communication Links, determine solutions to meet the revised timing requirements. This results in revised demands being placed on to Spectrum. Spectrum determines a solution that now satisfies both demands and notifies the Conflict Resolution component that the conflict has been resolved. Spectrum also confirms the successful resource allocation to both Sensors and Communication Links and the success of the planning activities is subsequently reported back up to Tasks.

A.2.9.8.2.2 Conflict Brokering (Planned Interruption)

In a variation of the previous scenario, the requirement placed by Communication Links is less definitive. Specifically, the exact timings for which the Communication Links component requires use of the spectrum resource is not yet precisely known. The requirement is therefore expressed as short interval durations, of unspecified timing, occupying a percentage of time within an extended time window. Consequently, as in the previous example, Spectrum determines that the time window for the subsequent Sensors spectrum allocation request creates a conflict with the existing (communication) allocation since it must default to the extended time window when considering the potential for overlap. Spectrum therefore notifies the Conflict Resolution component of the conflict.

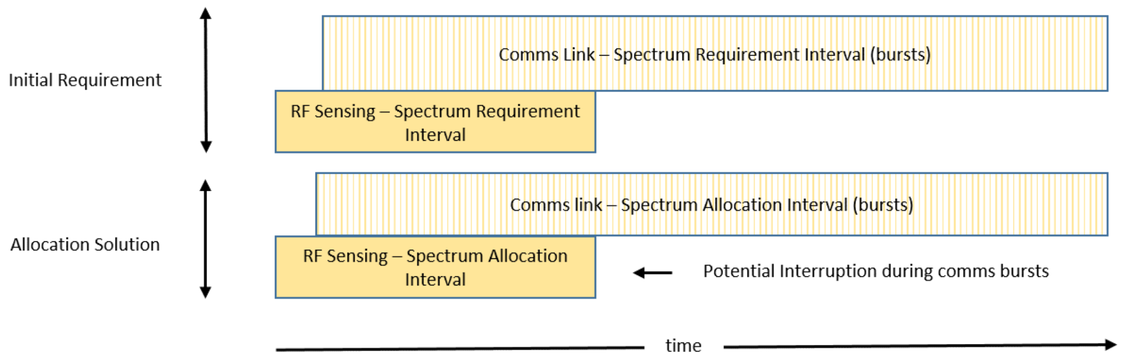

Initial Requirement
Comms Link – Spectrum Requirement Interval (bursts)
RF Sensing – Spectrum Requirement Interval
Allocation Solution
Comms link – Spectrum Allocation Interval (bursts)
RF Sensing – Spectrum Allocation Interval
Potential Interruption during comms bursts
time

Figure 72: Planned Interruption

The Conflict Resolution component determines that nothing within its configured rules warrants immediate arbitration and determines the brokering process that should be followed. In accordance with the defined brokering process, the Conflict Resolution component triggers both Sensors and Communication Links to seek to refine the relevant solution. In response, each component consults Spectrum to determine the refinement goal.

Spectrum confirms the refinement goal to Communication Links, which is to remove the spectrum requirement for the specified time period required by Sensors, but Communication Links is unable to find an alternative solution to meet the goal within the bounds of its existing requirements and constraints, and notifies the Conflict Resolution component.

Given the nature of the requirement placed by Communication Links, Spectrum identifies a refinement goal for Sensors to remove the spectrum requirement for an extended time window required by Communication Links, or to accept a defined probability of interruption during this window. Sensors is also unable to meet the refinement goals within the bounds of its existing requirements and constraints and notifies the Conflict Resolution component.

Conflict Resolution undertakes similar exchanges with Sensing and Networks, which results in the identified need for corresponding relaxations of their timing requirement.

Networks is unable to find an alternative solution since it has no flexibility in its timing requirement. Sensing is also unable to remove the sensor requirement for the entire time window specified, but determines that it can still complete its action, with an adequate probability of success, even with the potential interruptions.

Sensing therefore places a revised requirement on to Sensors (which allows for the potential interruptions), and Sensors in turn places the revised requirement on to Spectrum. Spectrum confirms the allocation and confirms to the Conflict Resolution component that the conflict has been resolved.

During solution execution, there are two possible outcomes:

- Sensors utilises the spectrum resource and the request for use from Communication Links occurs at a time where there is no conflict.
- Sensors is using the resource when the request from Communication Links occurs.

In the latter case, in accordance with its planned solution, Spectrum now allocates the resource for use by Communication Links and notifies Sensors that the resource is no longer available for its continued use. Sensors is notified when the resource becomes available for use again. Depending on the nature of the sensor and communications tasks, this may occur repeatedly over time due to the 'burst' nature of use for communications purposes, with brief windows of availability for the sensor interspersed in unused periods between transmission bursts. In both cases the process has enabled greater use of the resource than would have been achieved without the flexibility of allowing for interruption.

A.2.9.8.2.3 Conflict Arbitration (Unplanned Interruption)

In a further variation of the first scenario, a threat is detected during the search activity that requires the use of countermeasures; this scenario now includes the use of the Countermeasures and Effectors components. The required effector to support the countermeasure results in a conflict in spectrum requirements. The primary events of this scenario are highlighted in red in Figure 70: Resource Management Example 2.

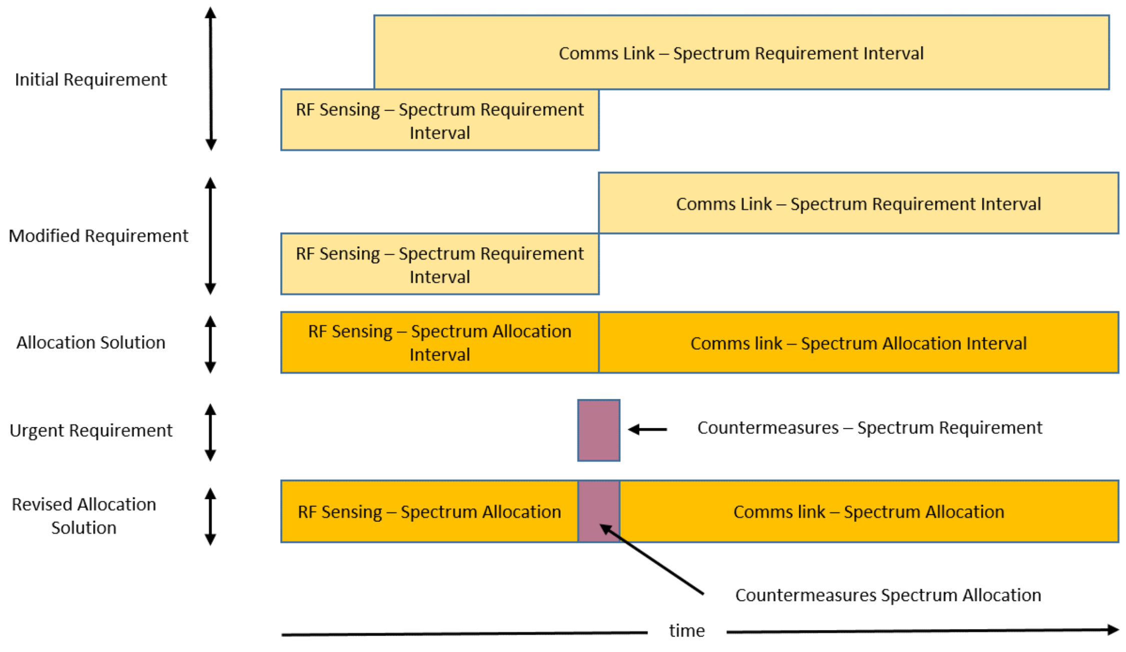

Initial Requirement
Comms Link – Spectrum Requirement Interval
RF Sensing – Spectrum Requirement Interval
Modified Requirement
Comms Link – Spectrum Requirement Interval
RF Sensing – Spectrum Requirement Interval
Allocation Solution
RF Sensing – Spectrum Allocation Interval
Comms link – Spectrum Allocation Interval
Urgent Requirement
Countermeasures – Spectrum Requirement
Revised Allocation Solution
RF Sensing – Spectrum Allocation
Comms link – Spectrum Allocation
Countermeasures Spectrum Allocation
time

Figure 73: Unplanned Interruption

Upon detection of the threat, Countermeasures is engaged to determine a response. The solution determined requires the use of a jammer and Countermeasures requests the immediate use of the required portion of the RF spectrum. Spectrum identifies that the request conflicts with an ongoing (sensing) allocation and an upcoming (communications) allocation and notifies Conflict Resolution of the conflicting requirements. The Conflict Resolution component assesses the conflicting requirements and determines, in accordance with its configured rules, that:

- The Countermeasures requirement matches criteria that make it of a critical nature requiring immediate arbitration (e.g. based on the origin, nature and priority of the request).
- The requirements do not trace to a common origin that could act as an arbitration authority and that it (Conflict Resolution) is the designated arbitration authority.

The Conflict Resolution component determines that the impact on Sensors and Communication Links is non-critical and informs Spectrum of the arbitration decision. Spectrum immediately re-allocates the resource for use by Countermeasures and notifies Communication Links and Sensors that the resource is no longer available for use. Communication Links and Sensors consequently provides upward notification on the status of their activities. Countermeasures engages Effectors to instigate the jamming response.

A.2.9.9 Resource Management Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Resource Management Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept identifies the following generic component behaviours in relation to resource management: • Components may generate requirements that directly, or indirectly, result in demand for resources. • Components support a process of brokering and arbitration, when resource conflicts occur, by: • Performing solution refinement to achieve a specified goal (aimed at resolving the resource conflict). • Determining solution refinement goals for higher-level components (aimed at resolving the resource conflict). • Making arbitration decisions where their dependent requirements produce resultant resource demands that directly conflict.	The PRA defines the following generic component responsibilities and services that enable components to directly or indirectly generate resource demands and to support the brokering and arbitration of conflicts: **Responsibilities:** • determine_solution • determine_solution_dependencies **Services:** • Requirement • Dependency_Refinement • Solution_Dependency For more information, please see the Dependency Management PYRAMID concept.
Specific PRA Component support for the PYRAMID Concept	The PRA separates out the following areas of functionality with regards to resource management: • Understanding of resource availability, the determination of solutions to satisfy resource demands and the identification of conflicts. • Understanding, and application, of the processes and rules for the brokering and arbitration of resource conflicts.	The PRA defines the following components whose role relates to resource management: • Resource Layer components. • Conflict Resolution The PRA defines the following generic component responsibilities and services that enable Resource Layer components to manage resources: **Responsibilities:** • capture_requirements • determine_solution • identify_conflict **Services:** • Requirement • Broker_Conflict For more information on the Resource Layer components, see the Control Architecture PYRAMID concept. For more information on the Conflict Resolution component, see Dependency Management PYRAMID concept.
Exploiter considerations	The following resource management related considerations are a matter for individual Exploiting Programmes: • The choice to manage a resource or not. • The methodologies for allocating and scheduling resource uses. • The level of granularity and precision to which resources are allocated. • The choice of if, how and when to cater for resource replenishment and contingency. • The specific rules and methodologies used for the brokering and arbitration of conflicts.	The PRA does not mandate a specific approach in respect of these considerations.

Table 10: Resource Management PYRAMID Concept Summary

A.2.10 Operational Support

A.2.10.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Control Architecture
- Constraint Management
- Autonomy
- Health Management
- Recording and Logging
- Human-Machine Interface
- Data Exchange
- Use of Communications

A.2.10.2 Introduction and Scope

This PYRAMID concept explains how the PRA can be used for purposes beyond the immediate operation of a mission. It describes a range of such uses, and explains the ways the PRA could be used and the components that would support such uses.

A.2.10.2.1 What is Operational Support?

"Operational support" includes any activities that support the mission but are not directly involved in carrying it out. These include activities which are:

- In preparation for a mission (planning, rehearsal and briefing);
- Following a mission (debriefing, post-mission data handling and replay);
- To maintain or improve capability (maintenance and support).

Other similar uses are not excluded. Training, for example, is out of scope of this PYRAMID concept, but many of the same solutions would apply.

Support for these activities could be achieved in two ways:

1. Through the use of a dedicated, separate system (for example a mission planning or maintenance system), which exchanges information with the operational system. This separate system may or may not be a PYRAMID deployment.
2. By including both the capability needed to support the mission as well as to carry it out within the operational system.

Within this PYRAMID concept the term ‘operational system’ refers to an air vehicle or a sub-set of an air vehicle. Although not explicitly covered by this PYRAMID concept, many of the principles that are applicable when referring to an operational system will also be applicable to an air vehicle simulator or other training devices.

Figure 74: Separate Planning System shows an example of a dedicated, separate planning system, as in option 1 above. A mission planner would use this to define a Mission Plan that would be uploaded to the operational system. The planning system could have been developed as a PYRAMID deployment, as in option 2.

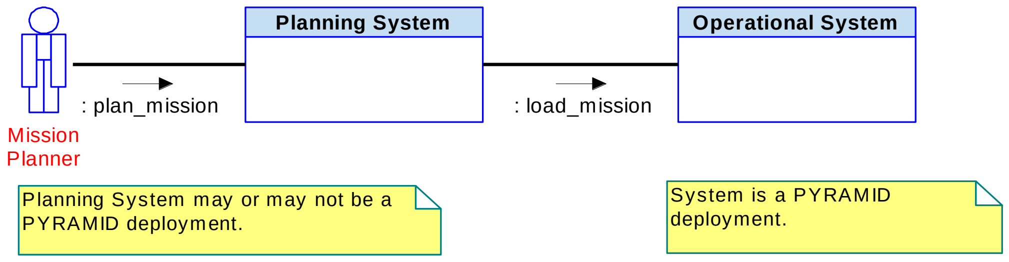

Planning System
Operational System
: plan_mission
: load_mission
Mission Planner
Planning System may or may not be a PYRAMID deployment.
System is a PYRAMID deployment.

Figure 74: Separate Planning System

Figure 75: PYRAMID Compliant Deployment used for Planning shows how a mission planner could use the operational system directly for mission planning. Mission planning requirements would be placed on the operational system to achieve this. Such a solution would exploit similarities between planning and operational uses.

Other support uses (briefing and debriefing, maintenance, etc.) would have similar options.

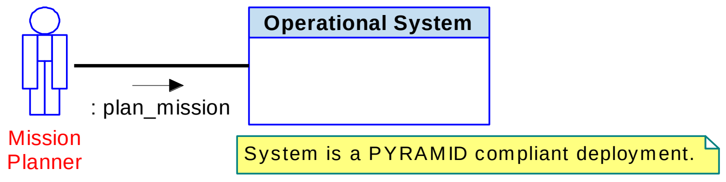


Figure 75: PYRAMID Compliant Deployment used for Planning

A.2.10.2.2 Why is Operational Support Important?

A PYRAMID compliant deployment will not exist in isolation. It will require information to be input prior to the start of a mission. Similarly, following the completion of a mission, information will need to be extracted to support functions such as debrief and maintenance and the collection of gathered intelligence.

This information exchange performed outside the execution of a mission is considered operational support. This PYRAMID concept shows how a PYRAMID deployment and other systems can be combined in a structured manner.

Without a well-defined interface to these other systems, the PYRAMID deployment may not be able to reliably perform the functions required of it.

A.2.10.3 Overview

All PRA components are responsible for the whole mission lifecycle, including planning, execution and post mission analysis. Component responsibilities therefore encompass all aspects of the mission lifecycle, in relation to their subject matter. While the PRA does not define any specialised components directly related to Operational Support, the following PRA components have roles that are especially relevant to aspects of Operational Support:

- Authorisation and Operational Rules and Limits, which allow the mission plan to constrain the operational system (see the Constraint Management and Autonomy PYRAMID concept);
- Information Brokerage and Data Distribution support the collation and distribution of mission information relating to mission data loading, and post mission analysis and briefing (see the Data Exchange PYRAMID concept);
- Health Assessment and Anomaly Detection, which support maintenance and (by enabling capability assessment) mission planning (see the Health Management PYRAMID concept);
- HMI and communication components, which provide the link with external systems and operators (see the Human-Machine Interface, Data Exchange and Use of Communications PYRAMID concepts).

The PYRAMID concept shows how PRA components can be used to provide a simulation harness for mission rehearsal or playback.

A.2.10.4 Component Composition

This PYRAMID concept identifies a number of generic component behaviours in relation to Operational Support. These behaviours are formally captured as component responsibilities in the PYRAMID Technical Standard, section Component Composition, along with associated component services. These responsibilities and services are also summarised in Table 11: Operational Support PYRAMID Concept Summary.

A.2.10.5 Mission Planning and Data Loads

Most PYRAMID deployments will rely on information that is changeable and specific for different uses and environments. Such information is contained in the Mission Plan, which comes with a large amount of supporting information needed by the operational system to carry out the plan.

The mission planning process will produce the Mission Plan for a flight of one or more air vehicles that will achieve a particular tasking order. Mission Planners will typically make use of a dedicated Mission Support System to create the Mission Plan, and this is the approach highlighted in this PYRAMID concept.

The information products output from this process then need to be transferred as loadable data into the operational system, to shape how it will operate in a given mission. The PRA facilitates this by allowing for components that are data-driveable, so that the architecture can be configurable, exploitable, and useable across a range of missions.

The provision of mission planning may be supported by or include simulation functionality (see Simulation & Playback Support section for further details).

A.2.10.5.1 Mission Planning Support

A dedicated Mission Support System, used by Mission Planners, is external to the operational system (as illustrated in Figure 74: Separate Planning System) and could either be a PYRAMID deployment itself (as shown in the Mission Data Load IV) that is separate from the operational PYRAMID deployment or a non-PYRAMID based procured solution. In either case, an external Mission Support System would generate the mission planning information products as loadable data, in a form that is tailored for the operational system, to be transferred through a communication channel or some other transfer mechanism. The Mission Data Load IV describes an example of how the Mission Planner could be assisted to ensure a complete set of Mission Plan data is prepared for loading using a PYRAMID deployment Mission Support System.

A.2.10.5.2 Operational Autonomy and Mission Planning Levels

The PRA supports the definition of desired autonomous behaviour for a system by explicitly representing activity authorisation and authorisation limits. In the operational support environment an operator will be seeking to define the desired behaviour or behaviour bounds. By limiting the system's authority to act and constraining the options available to the system, an operator can create a bounded problem space within which a system can plan and enact solutions autonomously without exhibiting undesirable behaviours.

Components will be granted authorisation within specific authorisation limits which control when an activity can be performed. More general constraints are handled by the Operational Rules and Limits component which imposes limits on other parts of the system in accordance with the rules given to it. These components are intended to be data-driven to allow them to be configured at run time with the desired limits and rules for an individual mission.

Depending on the level of autonomy an operational system is capable of, and programmed for by the Mission Planner, the Mission Plan may be in the form of objectives, tasks or actions, or a mixture. This is illustrated in Figure 18: Authorised Operator Interactions. For higher levels of operational system autonomy, the Mission Planner will interact with higher PRA layers of components. The layers of the PRA are described in the Control Architecture PYRAMID concept.

For example, for higher levels of system autonomy, where vehicles are given tasks to perform, mission planning would develop plans and supporting data to populate the Tasks component within the operational system, such that they have any required data-driveable content correctly populated. It would be expected that all PRA layers below the highest layer designated for autonomy would also require interaction from the mission planner via the Mission Support System. Regardless of the level of autonomy, the Mission Support System would always be expected to interact with components in the Resource Layer and some service components also, in order to develop the necessary resource specific vehicle fit data and operational level detail that ensures vehicle equipment and services can provide the capability and control needed to support the Mission Plan.

For further details on autonomy see the Autonomy PYRAMID concept.

A.2.10.5.3 Types of Mission Data and Update Frequency

Data to be loaded falls into two broad categories: engineering data and mission data.

Engineering data relates to internal processing employed by a component that is specific to a vehicle platform, or domain of operation. For example, the information required for determining vehicle performance will differ between fast jets, rotary wing, and sub-surface vehicles. Engineering data specific to a vehicle, vehicle type or domain of operation will not change frequently. Vehicle fit will also affect engineering data as different resources and equipment are swapped in and out between missions or potentially reconfiguration during a mission. The resources and equipment may be ballistic or advanced weapons; EM sensing technology options; or complementary countermeasure resources. This type of data load is carried out whenever vehicle equipment is changed.

Mission data relates to different theatres of operation, requiring, among other things, specific data to represent the geographic and tactical environments in which an exploitation will be deployed. For example, different missions will involve exposure to a range of different threats depending on the territory in which a mission is performed, as well as the purpose and characteristics of the mission itself. These threats and other entities are local to the mission territory, or the forces operating in the territory. Mapping and terrain data will also be specific to the mission. Mapping and terrain data is not likely to need re-loading for as long as a mission continues to be based in and around the same territorial boundaries and the data remains current, therefore this data is only likely to need re-loading when the mission footprint changes or updates are published. Some more detailed and specific mission tactical elements may need changing on a sortie-by-sortie basis in highly dynamic and evolving scenarios.

A.2.10.5.4 Data Load Interactions

Data loads are produced from mission planning activities that could be performed during a dedicated planning related mission phase or during operational system use, such as to support a re-planning activity. The size and related content of a data load therefore could vary from information to support an entire mission to an incremental update concerned with a single subject matter area, and anywhere in between, based upon the reason for the data load (including synchronisation of data) and information available at the time.

Data Load Manager

A Data Load Manager is responsible for the management and provision of any functionality to support data movement between instantiations of components or systems. This role could be an external actor (e.g. a human, a computer/ autonomous machine or a combination of the pair) or may be implemented using components such as Information Brokerage, dependent upon the implementation. The coordination of data loads is not performed by one particular component, and how this is achieved is Exploiting Programme specific. An example of this using the Information Brokerage component, with interactions from external actors managed via HMI (however the HMI aspects are not shown), is illustrated in Figure 187: Mission Data Load IV.

During planning, its role is to instruct applicable components to create the associated data load to support mission requirements.

During operational use of a system, its role is to inform components of the availability of new data, and to coordinate the mission data loading process.

The mechanism of data loading is related to the implemented mechanism of data transfer (e.g. communications or via data storage) and will therefore be Exploiting Programme specific.

Mission Data Load Creation

The range of interactions required leading to the creation of a successful data load are illustrated in Figure 187: Mission Data Load IV. In this specific example the Data Load Manager is represented by the Information Brokerage component and managed via HMI (although the HMI aspects are not shown). This includes the capture of data retention requirements by the Mission Planner, which is in line with the Recording and Logging PYRAMID concept. The Data Load Manager will determine and ensure that all information required for the chosen mission is drawn together to form the loadable package, which will then be transferred to the operational system once authorised.

Mission Data Load Loading

Successful transfer and data integrity during the loading process will be determined by the transfer channel. Once data loading has completed for all components identified across a platform requiring mission data, the receiving components as part of their fundamental set of responsibilities will check the data for insufficiencies and inconsistencies. Errors in component operation will occur if it is acting upon an incomplete or incorrect data set. It is the responsibility of the Data Load Manager to inform a component of a situation where it should suspend loading for a period, or to switch to a new data set.

The PRA supports the use of a variety of data loading mechanisms, including physical communication channels and physical movement of data storage media. The method of transfer will be Exploiting Programme infrastructure specific, however from a PRA perspective the interactions required for collation and movement of data do not differ between deployment design choices. Therefore the channel is considered independent of the transportation method.

Figure 76: Components supporting Mission Data Load Creation and Loading illustrates how component instances can be used in the planning and operational usage of the system, for the creation and collation of the Source Mission Load instructed by the Planning Data Load Manager, and subsequent coordination by the Operational Data Load Manager of the loading and distribution of the Target Mission Load.

Note: The transfer mechanism is not shown as this is transparent to the process being represented.

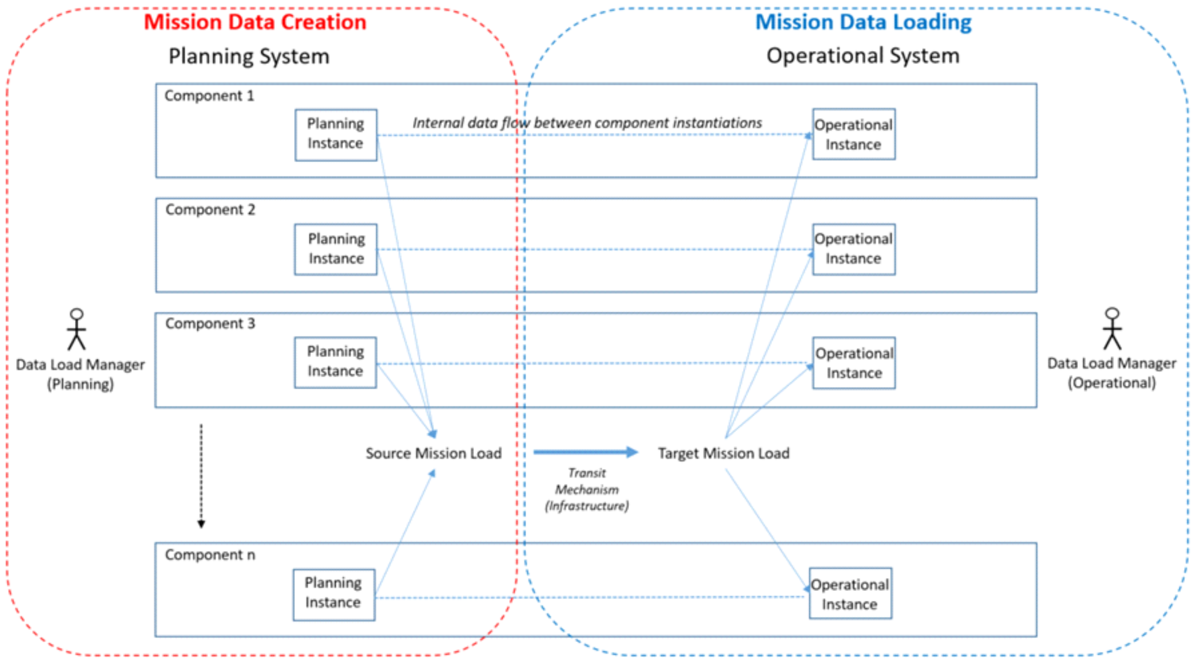


Figure 76: Components supporting Mission Data Load Creation and Loading

A.2.10.6 Mission Rehearsal and Briefing

Mission rehearsal and briefing are both part of the process of preparing for a mission. This section describes how they can be achieved using components as part of a mission planning deployment.

Mission rehearsal involves stepping through a mission plan to familiarise authorised operators with its content. Mission rehearsal may also have a role in validating the plan for operational use. Since no real actions take place during rehearsal, it will require a simulation harness (see the Simulation & Playback Support section) which replaces those components that interact with the real world with rehearsal versions. Most components will react to events as they are informed about them and will behave normally, unaware that they are in a simulation.

If mission rehearsal takes place within a separate mission planning system, the presence of simulation versions of components is unproblematic. The use of the operational system to provide simulation for mission rehearsal will only be a safe option if the implementation can clearly distinguish (for example by moding) when real and rehearsal versions of component are to be used.

The preparation of a mission report for briefing or other purposes is coordinated by the Information Brokerage component and collated by the Data Distribution component. The approach shown in the Generation of Handover Briefing IV will apply here.

A.2.10.7 Mission Debriefing and Post Mission Data Handling

Mission debriefing may occur during a mission, in support of crew handovers and mission awareness, or post mission for debrief and mission analysis. To support this an operational system may continually provide mission and intelligence reports directly following activities, or as required, to support situation awareness.

Post Mission Data Handling (PMDH) is the provision of mission generated data (including reports), in response to a query. While the originator of the query could be considered to be an external party, the post mission data analysis facility may be provided by a PYRAMID deployment (for example to support intelligence analysis), in which case the query and data provision would be from and to an ‘internal operator’.

Within the context of these two cases, a PYRAMID deployment must be capable of supporting:

- Varying levels of both information criticality and timeliness (e.g. in support of mission, post-mission and historical information exploitation).
- Both internal and external users.
- The provision of mission reports automatically, e.g. the generation of a mission report on completion of a task.
- The provision of mission information required in response to requests.

The provision of mission debriefing and PMDH can be facilitated by simulation functionality (see section Simulation & Playback Support for further details).

A.2.10.7.1 Levels of Information Criticality and Timeliness

The mechanism for the retention of data, including taking into account its required longevity, priority and criticality is detailed by the Recording and Logging PYRAMID concept.

For the purposes of debriefing and PMDH, information that is required to be available at a later point needs to be defined, its requirements understood and planned accordingly.

Data required for debriefing may be subject to a standard setting (for the operating unit) for criticality and timeliness, as may other mission generated data. Other operational data may be subject to a lower level of retention requirement, e.g. ‘retain if possible’.

A.2.10.7.2 Support for Internal and External Users

While the generation of data required by non-operational users is the same as for operational users, the delivery of such data to the users differs between those that are internal to the operational system and those that are external.

For example a PMDH user (whether internal or external to the system) may request data from the operational system, and receive an appropriately authorised response, in the same way that an operator could request data to gain situation awareness prior to handover. As explained in the Generation of Reports IV, the Exploiting Platform’s communications component suite would handle the generation and dissemination of reports and information to external users. In contrast, the Exploiting Platform's HMI component suite would manage internal user interactions. External users will have to be authorised to be able to interface with the system, and their method of interaction would be via their own HMI.

For further detail on use of the HMI components see the Human-Machine Interface PYRAMID concept. For further detail on use of the communications component suite see the Use of Communications PYRAMID concept.

A.2.10.7.3 Provision of Planned and Requested Mission Data

An operator, external user or Mission Support System may be automatically provided with generated data and reports as they are produced, or at pre-defined times as specified by mission requirements.

The Information Brokerage component has the reporting requirements, including what data, when it is required, and how it is to be delivered. These reporting requirements will be predefined, for example by the mission planner or as a standard set, e.g. for a specific mission type.

The Information Brokerage component will use these reporting requirements to marshal data and to inform an operator, via the HMI, what is available, allowing end user selection or automatic report generation as required. The reports will be constructed either in HMI or Data Distribution by collating the information from components that hold the source information required to populate the report.

Reports or mission data may be requested by a user, via either HMI or communications components as appropriate. An example of the request for, and delivery of, such data can be seen in the Request for Information IV.

Mission replay can be considered an example use of retained mission data. All data required for replay must be retained for reuse according to the reporting requirements held by the Information Brokerage component. The retrieval of such data must be achieved in a manner that allows for its use in the required order.

A.2.10.8 Maintenance and Support

Maintenance and support involves a wide range of activities to maintain the capability of a fleet, which are unlikely to be managed by a PYRAMID deployment. However, there will be significant communication between the maintenance and support system and the operational system so that the maintenance system will know the condition of the hardware being managed by the operational system, and the operational system can be updated with the post maintenance state of the assets.

It should not be assumed that the maintenance and support organisation will communicate with the operational system via a separate maintenance system. In small or lightweight organisations (including forward operating locations), management of maintenance and support may be manual, and the maintainer may communicate directly with the operational system.

The Health Management PYRAMID concept shows how the PRA can be used to identify anomalies in the deployed operational system and analyse the causes of such anomalies and (where they are caused by health conditions) to assess the consequences of a health problem. For the operational system, the focus of this activity is on the vehicle's capability and the consequences for the mission.

The support organisation has different needs, however. In particular, it is concerned with the identification of maintenance needs, such as specific Line Replaceable Units (LRU) that need repair or are reaching life or usage limits, and the state of consumables. It is still concerned with capability, but more at the fleet level where the capability of individual vehicles can be altered by carrying out repairs or changing their role fit.

Figure 77: Maintenance and Support shows the operational system communicating with a separate maintenance and support system. In both cases only a few components are shown: the operational system would have many more components concerned with planning and executing a mission, while the maintenance and support system would be responsible for all maintenance activities such as scheduling and logistics. In the diagram, both are shown as PYRAMID deployments, but for the maintenance and support system this might not be true.

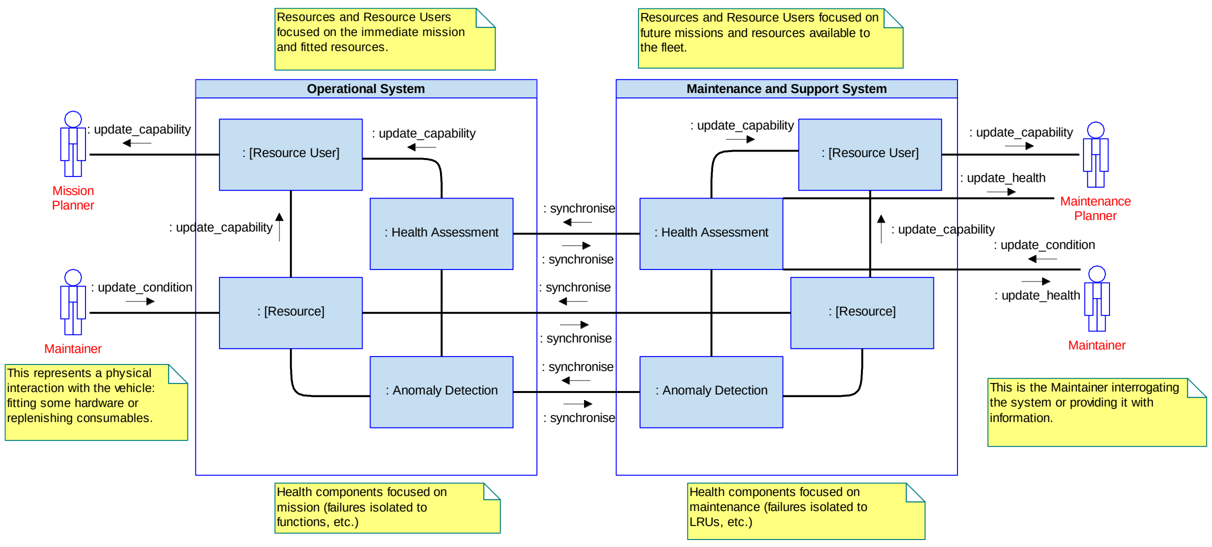

Resources and Resource Users focused on the immediate mission and fitted resources.
Resources and Resource Users focused on future missions and resources available to the fleet.
Operational System
Maintenance and Support System
: update_capability
Mission Planner
: [Resource User]
: update_capability
: update_capability
: Health Assessment
: synchronise
: synchronise
: update_condition
Maintainer
: [Resource]
: synchronise
: synchronise
: Anomaly Detection
: synchronise
: synchronise
This represents a physical interaction with the vehicle: fitting some hardware or replenishing consumables.
: update_capability
: [Resource User]
: update_capability
: update_health
Maintenance Planner
: Health Assessment
: update_capability
: update_condition
: [Resource]
: update_health
Maintainer
: Anomaly Detection
This is the Maintainer interrogating the system or providing it with information.
Health components focused on mission (failures isolated to functions, etc.)
Health components focused on maintenance (failures isolated to LRUs, etc.)

Figure 77: Maintenance and Support

In Figure 77: Maintenance and Support, [Resource] represents any component in the Resource Layer which is responsible for managing hardware. [Resource User] stands in for any component in the upper layers of the control architecture which depends on those resources to provide its capability. Health Assessment and Anomaly Detection components have the roles described in the Health Management PYRAMID concept.

Although superficially similar, the details of the two systems will be different because of their differences in scope and granularity. The operational system is focused on the immediate mission and fitted resources: the primary concern of health components is to determine the system’s capability so that the rest of the operational system can make informed decisions about the mission. The maintenance and support system is focused on future missions and resources that are available to the fleet. Health components need to identify failed or degraded LRUs, and those that have reached the end of their useful life (measured either by time or usage) so that maintenance can be planned. These two goals are consistent (both are descriptions of the same underlying health condition) but may require a different level of detail.

The Mission Planner interrogates the operational system to discover its capability. Other interactions of the Mission Planner are discussed in the Data Load Interactions section and are not shown here. The Maintenance Planner interacts with the maintenance and support system to plan a maintenance schedule which will maximise fleet availability as well as meeting requirements from mission planning. The Maintainer will replace LRUs, replenish consumables, and install role fit equipment as determined by the maintenance plan. The Maintainer may contribute to fault diagnosis by providing measurements or running tests. Where the Maintainer makes a physical change to the vehicle this will be reflected in both systems. For example, if the Maintainer adds fuel to the vehicle, this will be seen as a maintenance action in the maintenance and support system and also as a direct effect on the vehicle that will detect, via its measurement sensors, that its fuel contents have changed.

A.2.10.9 Mission Analytics and Performance Assessment

Mission analytics and the capture of data for performance analysis allows an operational system to determine its effectiveness and the effectiveness of the platform(s) it controls. Analysis may occur either during a mission or as a Mission Support System activity after a mission. However, the capability to carry out analysis on a platform during a mission allows changes to be made to planned and in-progress activities to incorporate the results of that analysis. It would also minimise the quantity of potentially classified raw data which needs to be retained and transmitted for analysis to a Mission Support System. The identification and retention of data needed is done on a component by component basis, taking into account its required longevity, priority and criticality, and is detailed by the Recording and Logging PYRAMID concept. This means that performance analysis could take place constantly at any level of the Control Architecture and can be captured at whatever level is required by a Mission Support System for a given purpose.

A.2.10.10 Authorisation

Operational support organisations have procedures for authorisation which can be complex and need to take into account factors such as the following:

- What is being authorised.
- When the authorisation is being provided.
- Whether supporting data has been authorised via its own authorisation chain; for example, Techniques, Tactics & Procedures (TTP’s) are developed (and authorised) by warfare centres remote from operational units, when authorising a mission plan at an operational unit the authoriser needs to be sure that any TTP’s the mission plan relies upon or implements are themselves authorised.

- The type of mission being authorised; some missions have specific objectives with detailed plans to achieve those objectives in which case authorisation is sought after. However a lot of missions are more loosely defined; the operator may be seeking to maximise flexibility in responding to whatever is encountered during the mission.

The Authorisation component understands the mechanisms for obtaining authorisations for proposed courses of action including where to obtain authorisation from, it thus supports the need to obtain different authorisations from different sources. It can also maintain a record of the authorisation associated with any data provided to the system. The relationship between authorisations will also be managed by the Authorisation component.

For information on granting authorisation see Autonomy PYRAMID concept, section Authorisation.

After a mission has been completed there will likely be a significant amount of data generated; whether and to whom to release this data will be something that needs to be carefully controlled via some form of authorisation process. Preparation of data for release is a combination of Information Brokerage which identifies the data, and Data Distribution which collates it. Whether data is authorised for release is a separate issue which will be handled by the Authorisation component.

A.2.10.11 Simulation & Playback Support

A system may need to contain a simulation facility to support pre-mission related activities such as mission rehearsal, briefing, planning and training, or a playback facility to support post-mission related activities such as data analysis and handling, de-briefing and training. These facilities would allow an operator to progress through a planned or completed mission via the manipulation of parameters such as time or location, and then observe the resultant system behaviour. Such facilities may be realised by simulation-specific instances of components.

A pre-mission facility would interact with components either functionally fulfilling their designed operational role, or a limited simulated role. A post-mission facility would interact with data recorded during a specific mission.

A.2.10.11.1 How does the PRA Support this Provision?

The components identified to provide simulated behaviour could be considered as part of a simulation or test harness. The purpose of this harness is to define the logical boundary between the normal operational components and the simulation specific components which are able to be manipulated based on the needs of the scenario. In addition to potentially performing some of their normal functions, components in the harness may also provide a simulated function, such as by giving a pre-defined response to an external stimulation. These components could report on the performance of the simulated activity without actually carrying it out; e.g. the Release Effecting component could be instructed to report the release of a store without actually releasing the physical store.

A.2.10.11.1.1 Simulation Provision

Figure 78: Example Simulation Harness Provision illustrates how a pre-mission rehearsal capability could be achieved using a set of components working concurrently to provide both real and simulated functionality. Components outside the simulation harness would be operating under normal mission conditions and assume that all information inputs are real. Components within the simulation harness

may be aware that they are operating under simulated conditions and provide simulated behaviour based on the needs of the scenarios.

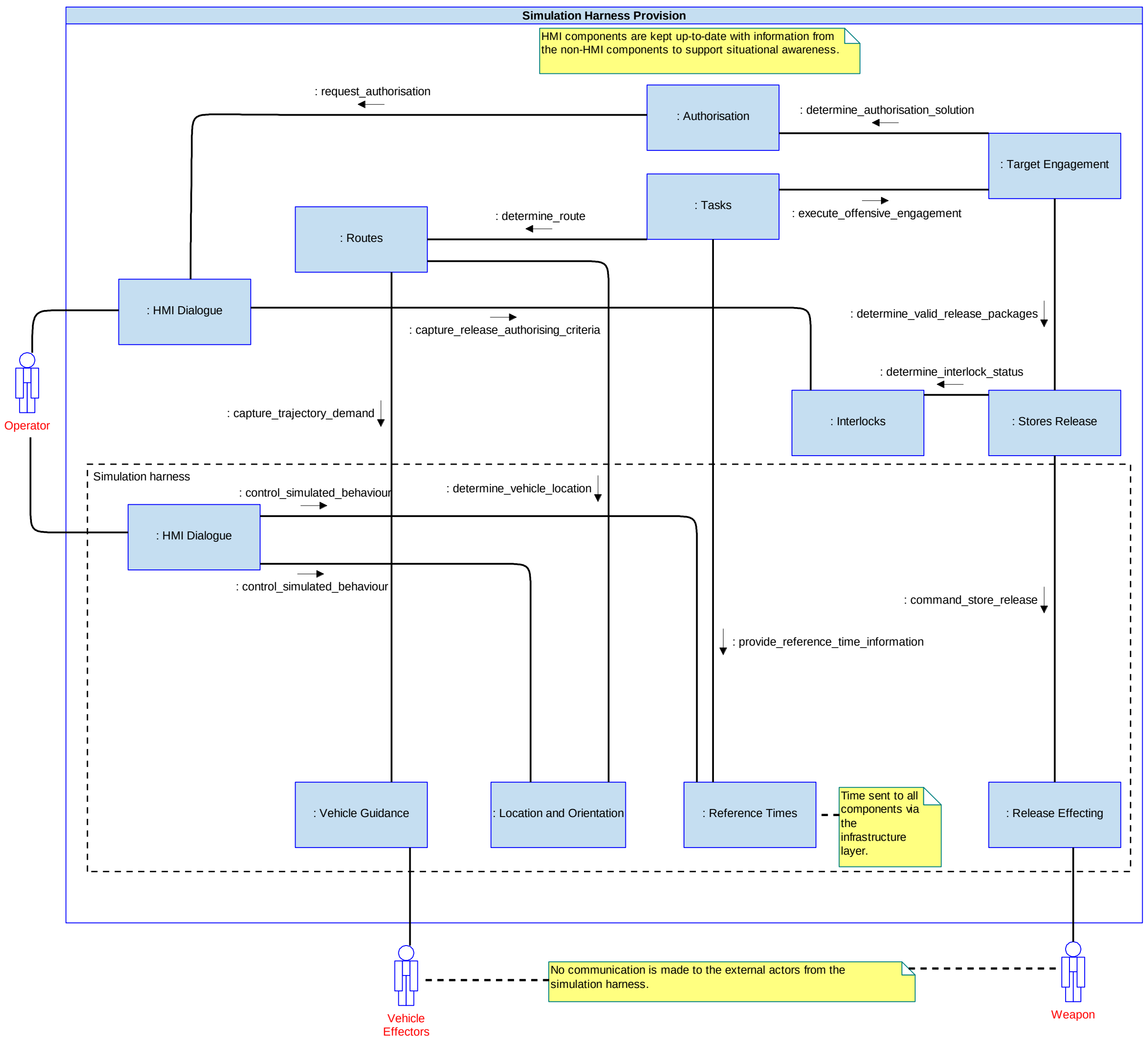

Simulation Harness Provision
HMI components are kept up-to-date with information from the non-HMI components to support situational awareness.
: request_authorisation
: Authorisation
: determine_authorisation_solution
: Target Engagement
: Tasks
: execute_offensive_engagement
: Routes
: determine_route
: HMI Dialogue
: capture_release_authorising_criteria
: determine_valid_release_packages
: determine_interlock_status
Operator
: capture_trajectory_demand
: Interlocks
: Stores Release
Simulation harness
: control_simulated_behaviour
: determine_vehicle_location
: HMI Dialogue
: control_simulated_behaviour
: command_store_release
: provide_reference_time_information
: Vehicle Guidance
: Location and Orientation
: Reference Times
Time sent to all components via the infrastructure layer.
: Release Effecting
No communication is made to the external actors from the simulation harness.
Vehicle Effectors
Weapon

Figure 78: Example Simulation Harness Provision

In this example, the instances of:

- The Location and Orientation component would provide values associated with locations, and the Reference Times component time could provide deviations from the time source, as instructed by the simulation HMI (examples include jumping to a specific time or location in the planned mission, or playing through a planned mission at double speed).
- The Vehicle Guidance and Release Effecting components report back to the Routes and Stores Release components respectively that they have enacted any actions asked of them without actually fulfilling them, e.g. report that movement demands are being enacted or releasing of a weapon without actually commanding the engine or weapon release.

All other components operate as per their normal operational behaviour.

A.2.10.11.1.2 Playback Provision

To support post mission based capabilities, the simulation HMI could provide a playback facility which would allow operational system recorded data to be retrieved from the system and presented back to the operator via the operational HMI or dedicated simulation HMI.

A.2.10.11.2 Deployment Considerations

Depending upon the specific requirements associated with a PYRAMID deployment:

(i) The boundaries of the simulation harness (thus the components encompassed within the harness) would change dependent upon the scope and level of simulation required to be provided.

(ii) Its simulation provision can be provided by defining the components as either:

- **multi-mode** (a component's internal behaviour is moded based upon whether the deployment is being operated in real-time or simulated), or
- **single-mode** (specific variants of components being created to perform either real-time or simulated behaviour) operation.

A.2.10.12 Operational Support Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Operational Support Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	All PRA components are responsible for the whole mission lifecycle, including planning, execution and post mission analysis. Component behaviours therefore encompass all aspects of the mission lifecycle, in relation to their subject matter, including: **Pre-mission:** • System maintenance and support • Data loading • Mission plan generation • Simulation/rehearsal • Briefing **Operational:** • Data capture for mission analysis • Mission analytics and performance assessment **Post-mission:** • Post mission data handling • Mission analytics and performance assessment • Replay/debrief • System maintenance and support All components are responsible for validating received information (e.g. completeness and consistency of data load), where the scope of validation checking is commensurate with their subject matter knowledge and reasoning. All components are responsible for undertaking performance assessments commensurate with their subject matter knowledge and reasoning, e.g. as part of solution optimisation, or to provide an assessment of expected or actual performance during and post mission.	All PRA components support the different aspects of operational support, related to their subject matter, through the following facilities and mechanisms: • Different instances or variants of components may be used to perform specific functions (within their subject matter) that are associated with different aspects of the mission lifecycle, e.g. different variants (corresponding to the same PRA component) may be used as part of a mission support system and on an operational platform. Equally a single component variant may incorporate the capability to support multiple, or all, lifecycle phases. • In principle, all PRA components can support moding that would make them suitable for use in a variety of operational support systems, e.g. in planning, simulation or rehearsal applications, as well as during operational mission usage. • All PRA components provide, and may retain, information relating to their subject matter. The PRA defines a number of generic component responsibilities and services in relation to recording and logging, and storage, that enable the capture of data for operational and post mission activities. This includes health information that may be used to help maintain system capability and support system maintenance/support. For more information see Recording and Logging, Storage, Health Management and Capability Management PYRAMID concepts. • In principle, all PRA components may utilise data driving, enabling support for data loading and mission specific configuration of components. While the appropriate use of data driving is recognised as good practice, the PRA does not mandate its use nor defines any associated services. • All PRA components support the loading and extraction of mission data and engineering data in accordance with their subject matter. The PRA defines the following generic component responsibility in relation to validating received information (regardless of the source of the data): • data_validation The PRA defines the following generic component responsibility in relation to performance assessment: • determine_quality_of_deliverables (For more information see the Dependency Management PYRAMID concept).
Specific PRA Component support for the PYRAMID Concept	The PYRAMID concept does not identify any specialised subject matters in relation to operational support.	While the PRA does not define any specific components directly related to operational support, the following PRA components have roles that are relevant to aspects of operational support: • Information Brokerage and Data Distribution support the collation and distribution of mission information relating to mission data loading, and post mission analysis and briefing (see the Data Exchange PYRAMID concept). • Authorisation and Operational Rules and Limits, allow the mission plan to constrain the operational system (see the Constraint Management and Autonomy PYRAMID concepts). • Health Assessment and Anomaly Detection, support maintenance and (by enabling capability assessment) mission planning (see the Health Management PYRAMID concept). • HMI and communication components, provide the link with external systems and operators (see the Human-Machine Interface, Data Exchange and Use of Communications PYRAMID concepts).
Exploiter considerations	The following operational support related considerations are a matter for individual Exploiting Programmes: • System partitioning to support different operational support activities, e.g. the implementation of dedicated systems for mission planning, data loading and extraction, simulation, and maintenance. • The approach to using multiple instances of components, component variants or moding, to support different mission phase activities. • How or when to apply data driving and mission data.	The PRA does not mandate a specific approach in respect of these considerations.

Table 11: Operational Support PYRAMID Concept Summary

A.2.11 Storage

A.2.11.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Recording and Logging
- Security Approach

A.2.11.2 Introduction and Scope

This PYRAMID concept describes the mechanisms provided by the PRA to control storage of data, and the interactions of components with storage facilities provided by an Exploiting Platform. The PYRAMID concept will address how a deployment may facilitate the retention of the data, and the support that the PRA provides in achieving that.

Whilst the general principles of this PYRAMID concept should apply to the method of storage and the storage media used by the deployment software and other programmable content present within a deployment, the PYRAMID concept has been written from the point of view of the data used for, and arising from, a mission. An Exploiting Platform’s software storage, its security requirements and means of protection are subject to the requirements of the Exploiting Programme as defined by the appropriate design authority.

A.2.11.2.1 What is Storage?

Within this PYRAMID concept, “storage” always refers to the action or method of storing data for future use, and not the component (which is always referred to as “the Storage component”) or the storage media (which is always referred to as “storage media”). For the purposes of this PYRAMID concept only, "data" can be considered any form of content that can be stored on storage media available to the deployment.

Considerations relating to storage can be separated into the following areas:

- Storage to media - writing the data to the appropriate storage media.
- Security domain partitioning - separation of different types of data items in order to satisfy the security objectives for confidentiality, integrity and availability, etc.
- Data retention - the mechanism by which important data is kept for future use or reference.
- Temporary data storage for facilitating processing needs.
- Deletion and sanitisation - the disposal of data from storage media.

Note:

- The determination of the need for data retention by a component is covered within the Recording and Logging PYRAMID concept.

- It is assumed that the system will work on the premise that the data will be passed by value rather than by reference.

A.2.11.2.2 Why is Storage Important?

It is expected that any Exploiting Platform will require and may generate large amounts of data during its operation. Some Exploiting Platforms may have capability constraints for storage such as limited storage capacity and others may have mandated requirements for storing certain items of data during operation, e.g. as required to meet specific legislative needs.

These requirements may change or evolve during the life of the Exploiting Platform, and it is therefore essential that the PRA is flexible enough to allow the Exploiter to choose the methods of storage, the management of storage media, and the data held.

The different types of data and their use will determine the retention needs, the storage methods, and storage media that will be used.

A.2.11.3 Overview

This PYRAMID concept can be summarised as follows:

- Components will require access to storage media, with this access being agnostic of the storage infrastructure.
- The Storage component manages the storage media, including the need for separation, encryption, sanitisation, etc., but excluding the actual transfer of data between components and storage media.
- Components are responsible for accessing, storing and retrieving their own data. The Storage component may apply constraints on the amount of data that can be stored.

A.2.11.4 Component Composition

This PYRAMID concept identifies a number of generic component behaviours in relation to Storage. These behaviours are formally captured as component responsibilities in the PYRAMID Technical Standard, section Component Composition, along with associated component services. These responsibilities and services are also summarised in Table 12: Storage PYRAMID Concept Summary.

A.2.11.5 Storage Considerations

A.2.11.5.1 Uses of Storage

A component may need to store data for either temporary storage or retention purposes.

Temporary storage covers where a component uses storage in order to facilitate its immediate processing needs, and execution of its responsibilities. For temporary storage, the component knows why it needs it and for how long. For example any component may have an allocated space in storage media (a memory cache) for discharging their processing requirements.

Retention storage covers where a component holds data for future use, possibly outside of the specific component instantiation's understanding, e.g. for report dissemination or post mission analysis. The determination of the retention strategy for a component's data is covered within the Recording and Logging PYRAMID concept.

A.2.11.5.2 Storage Media Types

An Exploiting Programme will choose the types of storage media it needs. Different types of storage media may be used to satisfy a component's different data storage needs, such as those relating to data access time, longevity, or susceptibility to corruption.

A.2.11.6 Component Interaction with Storage Infrastructure

A.2.11.6.1 Component Storage Access

The data physically stored on storage media, for either temporary storage or retention storage needs, is considered to be outside of a PYRAMID component. Despite this, the data is still owned by a single PYRAMID component, since it is either within the subject matter of the associated PRA component definition or is used to log the components activities, and so is ultimately its responsibility to manage what is and is not stored.

The mechanisms by which components can access data (meaning read, write, and delete) will typically be hidden from the PYRAMID component design. The PYRAMID component will interact with elements of the execution platform, such as the operating system, which will manage the details of the data access, including the physical state changes of the storage media. Therefore, the act of storing data (how and where data it is physically stored) is typically hidden from the PYRAMID component design.

It is important that the Exploiter ensures that the PYRAMID component is robust to, and can manage, the impact of unexpected changes to its stored data, such as the loss of availability of data held on storage media.

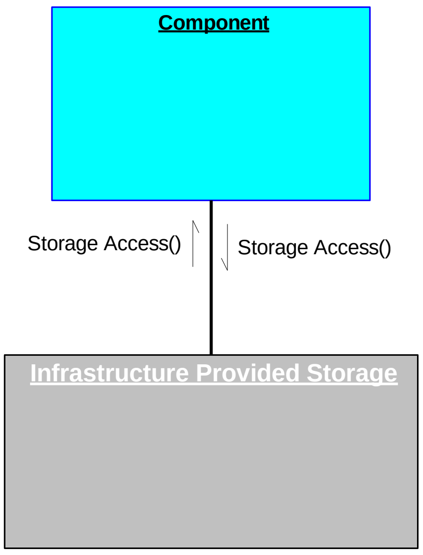


Figure 79: The Act of Storage

Additional notes:

- A component's data may need to be loaded before it can be retrieved. Management of data loading is detailed in the Operational Support PYRAMID concept.
- The Exploiting Programme may require stored data to be encrypted, see the Encrypted Storage Media section.
- The above description has been written on the assumption that the computing hardware has an operating system. However, if the component is the only software hosted on computing hardware without an operating system, it will have a greater awareness of the storage media and a more direct interaction with it.

A.2.11.6.2 Access to a Component's Data by Other Components

Components cannot access storage media data belonging to other components directly. If data needs to be communicated from one component to another, this should be provided by a service of the component that owns the data.

A.2.11.7 Management of Storage

While components are not ‘aware’ of the storage infrastructure that they may be using, it may be necessary for a deployed system to be able to manage the capacity and use of the storage media provided by the infrastructure. Management of the storage infrastructure is the role of the Storage component. The use of the Storage component is more likely when storage across different systems (such as an air vehicle and a ground station) or significant system partitions needs to be managed, as opposed to directing routine storage operations commonly handled by the infrastructure.

While all components may indirectly access the storage media, the Storage component is responsible for interacting with the storage infrastructure (i.e. the storage media resource) in order to manage the available storage media capability. This includes ensuring that actions are taken if storage media capability is limited.

The Storage component will maintain a view of the usage of storage media, as illustrated in Figure 80: Storage Component View of Storage Use, and provoke the disposal of data items in line with the data storage policy of the Exploiting Platform. Identifying the data storage requirements and policy are Exploiting Programme considerations and are therefore the responsibility of that programme.

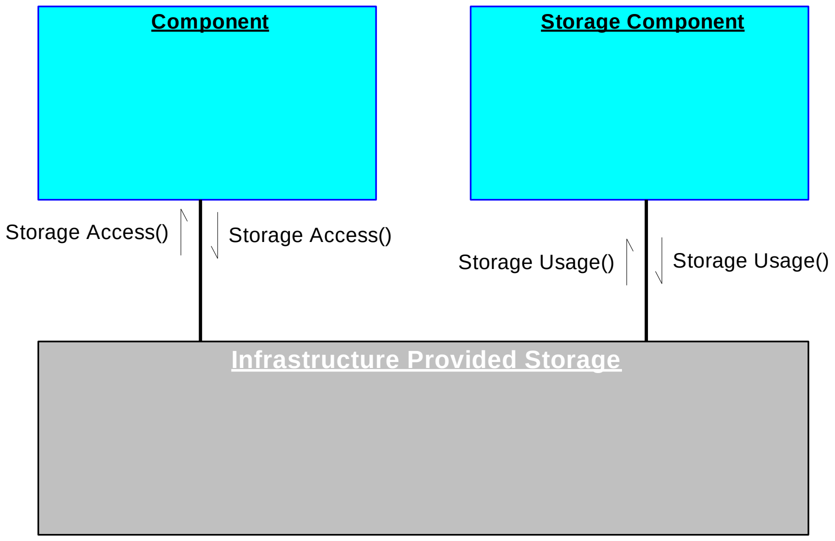


Figure 80: Storage Component View of Storage Use

Corrective action may be via deletion or sanitisation of data, or another storage infrastructure-specific option (e.g. movement, resizing or reallocation). The Storage component holds the knowledge of the data storage policies and will select the most appropriate action based on a policy and several factors such as:

- Overall storage capacity.
- Current remaining storage capacity.
- Storage media usage (e.g. ownership of data).

The Storage component manages the application of security to storage media, if required, e.g. encryption of drives (see the Cryptographic Management IV for an example of how encryption may be applied). The application of a component's data retention policy is covered in the Recording and Logging PYRAMID concept.

A.2.11.7.1 Movement of Data between Storage Media

The movement or duplication of data between storage media devices is performed by the infrastructure; however, this may be carried out under the direction of the Storage component or other parts of the infrastructure.

Where two instances of the same component exist, such as for redundancy purposes, data may need to be passed or synchronised between these instances, resulting in movement or duplication of data between storage media devices (or areas of memory on a device). However, from the perspective of these component instances they are just outputting and inputting data, with no knowledge of how the data is transferred, as well as potentially requesting their own data to be stored or deleted (as described in the Component Interaction with Storage Infrastructure section). Therefore, there is no specific request from either instance to move or duplicate data between storage media devices.

A.2.11.8 Changes Affecting Storage and Data Retention Capabilities

A.2.11.8.1 Types of Change

Certain circumstances may result in the need to remove data from the storage media or to render all data held upon it permanently inaccessible. How this is done in practice is a consideration for the Exploiting Programme to determine, however this PYRAMID concept considers the following options:

- Deletion - removal of data by a component.
- Sanitisation - the process of deliberately, permanently and irreversibly removing or destroying programmable content or data to make it unrecoverable.
- Addition or removal - changes to the availability of storage media.

A.2.11.8.2 Deletion

A deletion is where access to a specific data item or a set of data is no longer retained. Because deletion relates to specific data, this is controlled by the specific component owning that data. Each component manages and performs deletions of its own data, whilst respecting any requirements for security logging, etc. (see the Recording and Logging PYRAMID concept for further details).

This PYRAMID concept considers a deletion to be driven by a change in the retention policy for that data. For example, if the Storage component requires a reduction of storage use as the available space limits are nearly reached, it could set a constraint on other components to instigate a change of the appropriate retention policies, meaning that those components subsequently delete the data no longer required from the storage media. This is shown in Figure 81: Storage Use Triggered Change of Retention Policy. Further information on retention policies can be found in the Recording and Logging PYRAMID concept.

How the components manage the application of the retention policy is internal to the component, and how knowledge of the volume, location and component ownership of stored data is maintained, by Storage, are decisions made by the Exploiting Programme (e.g. through use of metadata or a Table of Contents). Neither are covered in the PRA.

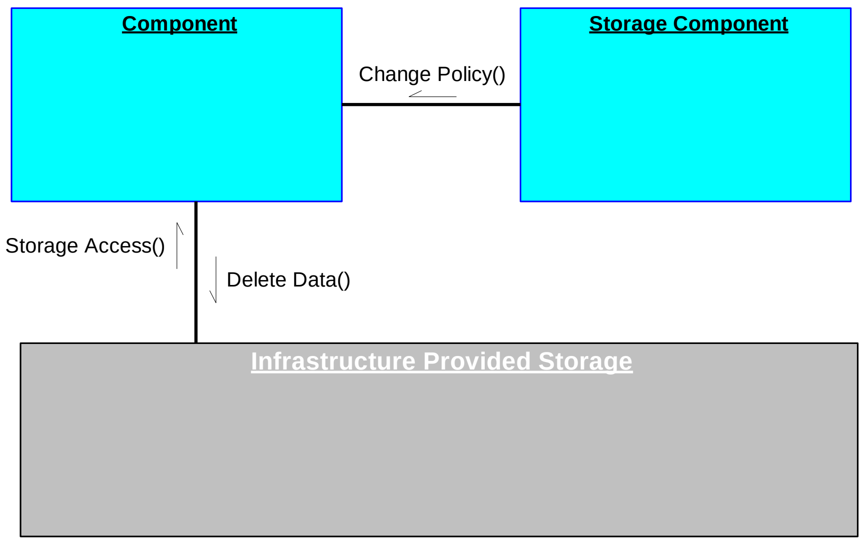


Figure 81: Storage Use Triggered Change of Retention Policy

A.2.11.8.3 Sanitisation

Sanitisation is the process of deliberately, permanently and irreversibly removing or destroying the data held on storage media to make it unrecoverable. Some forms of sanitisation will allow reuse of the storage media, whilst others are destructive in nature and render the storage media unusable. The scope of sanitisation may depend on the method used, but could be a specific data item, a partition, a single storage media device or some or all storage media used by the system. Within that scope, sanitisation is a blanket ruling, and no exclusions can be applied. For this reason, data should be carefully segregated and there may be specific requirements for ongoing retention of safety or security-related logs.

Sanitisation of data by the infrastructure may be initiated by the Storage component. The trigger for this to occur may be from operator request, or by another component determining a need, e.g. on entering an emergency situation. To ensure speed of destruction, it is recommended there is no component interaction or negotiation that may prevent sanitisation.

The Storage component directly interfaces with the infrastructure to direct sanitisation to occur, as shown in Figure 82: Storage Initiation of Sanitisation. It is the responsibility of the infrastructure to provide a mechanism to sanitise all of the data from the designated storage media.

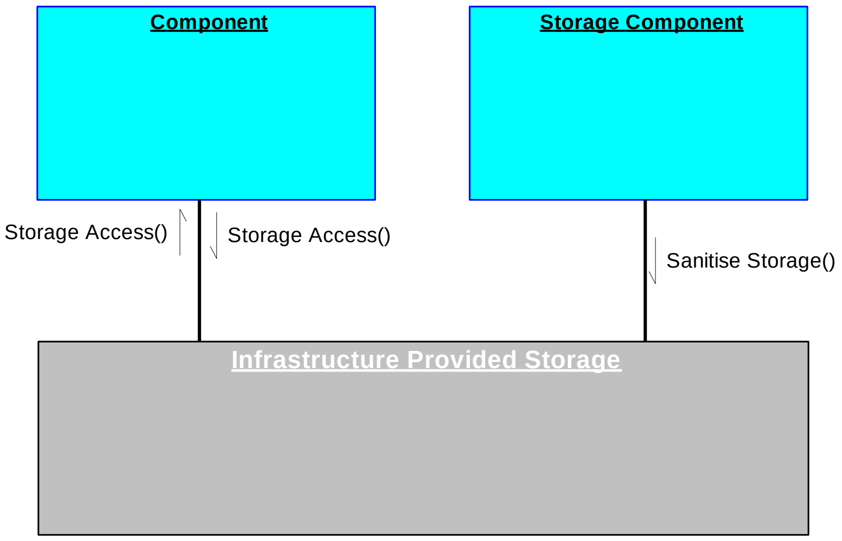


Figure 82: Storage Initiation of Sanitisation

When the Storage component is instructed to initiate sanitisation of the storage media, the Storage component may inform components so that their retention policies can be updated accordingly. This may be required to ensure that no further data is stored, or to manage any potential change of capability that a component may have from a change of data.

A.2.11.8.4 Change in Storage Media

When there is a change to the storage media available to an Exploiting Platform, for example when a new store is connected or an existing one is removed, the Storage component may be informed of this change by the infrastructure (see Figure 80: Storage Component View of Storage Use).

A change in storage media may lead to a change of capability due to data available to the components.

Where storage media is moved, for example node-to-node transfer of data on a USB drive, the impact of the change of availability of data is the responsibility of the components. Examples of data movement, e.g. data loads, can be found in the Operational Support PYRAMID concept.

A.2.11.9 Secure Storage

This section will discuss how the differing methods of storage support for security are provided by the PRA.

In order to achieve accreditation an Exploiting Programme will need to implement physical separation of storage media for at least:

- Original equipment manufacturer/integrator storage (e.g. for application software, in order to protect intellectual property).
- The storing of cryptographic material.
- Bulk storage media for access by the user.

This PYRAMID concept only explicitly covers the latter.

The Security Approach PYRAMID concept provides information on security domains and security features that are fundamental to the concept of secure storage.

A.2.11.9.1 Security Domain Partitioning

Security domain partitioning is the separation of data for reasons such as (but not limited to) differing security classifications in order to satisfy the security objectives for confidentiality, integrity and availability, etc.

This is often achieved using the following methods:

- **Physical Separation of Storage**, such as by implementing individual storage for instances of security classification, for example having two separate storage media devices, in order to securely separate data within different security domains.
- **Logical or Virtual Separation of Storage**, e.g. preventing access to parts of the storage media by access controls or encryption.

It is the responsibility of the Exploiting Programme to ensure that correct security provisions have been made and that data is written to the correct, appropriately security accredited, storage media and to manage gateways, etc.

A.2.11.9.2 Separation of Storage

It is expected that an Exploiting Platform will include a number of storage media devices. This will be determined by the needs of the Exploiting Programme. For example, there may be a requirement to isolate particular data, or to store certain data on a specific storage media device (such as a CSMU) in order to support flight certification requirements.

Separation of storage for security domain partitioning will support an Exploiting Programme's security accreditation. Separation of storage may also be used to support the design for safety, supportability and scalability.

A.2.11.9.3 Encrypted Storage Media

Encryption (and decryption) may be applied to the data being stored. Just as components are typically not aware of how and where data is being stored, neither are they aware of any encryption or decryption prior to storage or upon retrieval. The Cryptographic Methods component (supported by the Cryptographic Materials component) provides the encryption and decryption capabilities.

The management of such cryptography to provide encrypted storage is seen in the Cryptographic Management IV.

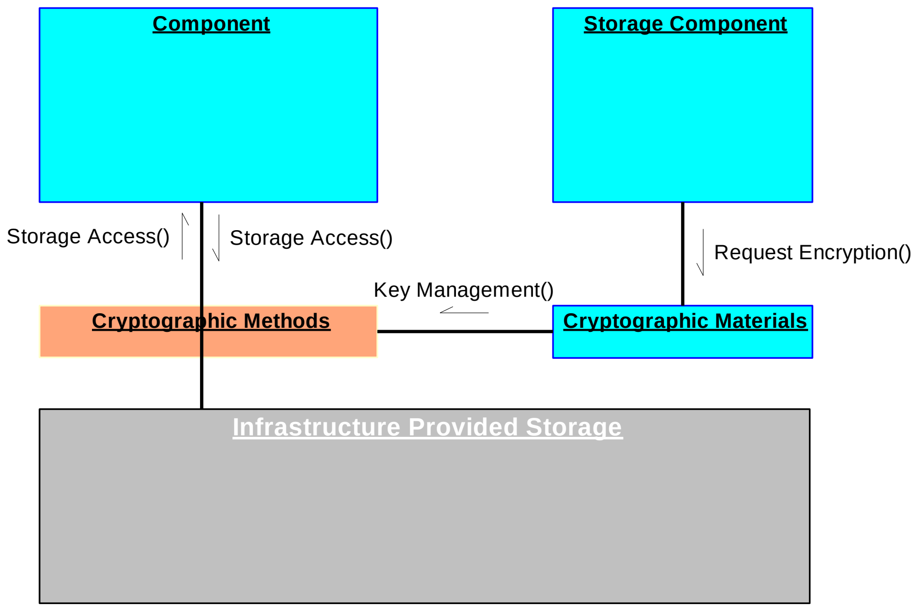


Figure 83: The Act of Encrypted Storage

Here, if the required encryption/decryption has not been enabled, the storage media, and thus any data therein, would be considered as unavailable for use.

Therefore, where appropriate, a component should be designed for robustness to prevent erroneous or inappropriate behaviour due to (possibly intermittent) unavailability of data on the storage media. This should be considered at the design stages of an Exploiting Programme.

In such cases, protection may be provided by ensuring that the Storage component is aware of the viability of the storage media for use: see the Change in Storage Media section of this PYRAMID concept and the Cryptographic Management IV.

Data stored on encrypted media can also be made inaccessible by revoking the cryptographic keys associated with the storage media (see Cryptographic Management IV). There are different types of cryptographic keys used and different areas of storage media can have different cryptographic keys. Sanitisation of the related cryptographic material will render the encrypted data in the storage media inaccessible (preserving confidentiality) whilst allowing for the recovery of data if the keys are later re-applied.

A.2.11.10 Storage Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Storage Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept identifies the following general component behaviours in relation to storage: • Individual components are responsible for determining the storage dependency requirements to meet their data retention and processing needs. • Individual components are agnostic of the storage infrastructure and have no responsibilities relating to the management of storage infrastructure (with the exception of the Storage component - see below). • Individual components are responsible for storing, retrieving and deleting their own data in accordance with their data retention strategy. • Components cannot access storage media data belonging to other components directly. Components access another components data via the services provided by that component. • Where it is necessary to replicate/synchronise data between PYRAMID component variants or instances that are derived from the same PRA component, this is behaviour that is considered internal to the PRA component. Individual components do not control the replication/synchronisation of data through transfer between storage media. • Individual components are responsible for addressing any shortfalls in their own capability resulting from the inability to store, retrieve or delete data.	The PRA defines the following generic component responsibilities and services in relation to storage: **Responsibilities:** • determine_storage_requirements • manage_data_retention_and_storage • address_capability_issue **Services:** • Storage_Dependency For more information on related responsibilities and services please see the Capability Management PYRAMID concept.
Specific PRA Component support for the PYRAMID Concept	The PRA separates out the following areas of functionality that relate to storage: • The management of storage infrastructure and storage space available to the system. • The process of applying cryptographic transformations (that may be used to encrypt and decrypt stored data).	The PRA defines the following components whose roles relate to storage: • Storage • Cryptographic Methods • Cryptographic Materials
Exploiter considerations	The following storage related considerations are a matter for individual Exploiting Programmes: • The type and scale of storage media and the policy for stored data management, e.g. deletion and sanitisation. • Methods for partitioning and protecting storage to achieve security and safety requirements. This includes methods for ensuring resilience of storage (e.g. redundancy) and component design for resilience to the loss of storage availability. • How knowledge of the volume, location and component ownership of stored data is maintained, by the Storage component.	The PRA does not mandate a specific approach in respect of these considerations.

Table 12: Storage PYRAMID Concept Summary

A.2.12 Recording and Logging

A.2.12.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Storage
- Security Approach

A.2.12.2 Introduction and Scope

This PYRAMID concept provides the framework for a retention strategy by which a component can identify items of data that are needed for future use and hence which need to be retained by recording or logging. It also has guidelines for the kind of rules for data retention within that strategy, such as how long a data item is required for and when a data item is no longer required.

It does not address the specific means by which the storage of these items is managed (this is in the Storage PYRAMID concept), or how it is published by the Information Brokerage component.

A.2.12.2.1 What is Recording and Logging?

A.2.12.2.1.1 Recording

Recording is the process of identifying and retaining data items directly associated with a component's subject matter, that need to be retained for future use (e.g. authorisations received). As the actual storage and publishing of items of data is handled outside each component, the component will need to identify the data that needs to be retained (as per its retention strategy) and, based on the details of its retention strategy, it will need to articulate the properties for how it should be retained, e.g. security level and integrity/robustness level.

Note that any data visible to a component can be subject to recording, regardless of whether it was generated by that component or the overall system within which that component resides (e.g. a communications component may record communications which are relayed through a platform as well as those for which it is either originator or recipient).

A component could therefore record data that it cannot make complete use of; for example, a communications component would be aware of data being transmitted and could therefore record the data and its intended destination, but if the data represents a specific graphical or audio format, the component could not actually understand the data it has recorded (but it would know the intended recipient).

A.2.12.2.1.2 Logging

Logging is the process of identifying and retaining data items associated with a component's processing that are not specific to its subject matter (such data items may therefore be common across components). Examples of this are computer runtime diagnostics data such as storing data for debugging, logging errors, updating internal data tables and logging events for non-repudiation, audit or other security-driven needs. As storage and publishing of data items is handled outside the component, the component will need to identify the data that needs to be retained (as per its retention strategy) and, based on the details of its retention strategy, it will need to articulate the properties for how it should be retained, e.g. security level and integrity/robustness level.

A.2.12.2.1.3 Recording and Logging Commonality

A single retention strategy, a set of specific retention rules, can be used to cover both the recording and the logging for a specific component. The process of identifying whether a data item has to be retained and applying a set of retention rules to it is identical regardless of the ultimate use of that data.

A.2.12.2.2 Why is Recording and Logging Important?

All Exploiting Platforms will have a general responsibility to recognise and select for retention the certain types of required data, and to know what data has been captured in this manner.

Recording allows a component to retain identified data items that are directly associated with its subject matter for future use.

Logging specifically allows a component to log and later articulate data about its current and past internal processes that may not be needed as part of its principle role but may be needed to meet its other responsibilities.

Examples of the need for Recording and Logging include retention of data for:

- Data required for mission use.
- Data for analysis of aircraft life and usage, e.g. structural stress data.
- Data for fault investigation, e.g. health and usage monitoring.
- Mission-generated data, e.g. for mission replay.
- Conforming to legislation, e.g. Crash Survivable Memory Units (CSMU). Note that General Data Protection Regulation (GDPR) requirements may constrain what can be stored.
- Data for non-repudiation and auditing purposes.
- Anomalous behaviour, event and message traffic data for cyber threat analysis.

A.2.12.2.2.1 Complying with Legal and Regulatory Frameworks

Data retention requirements may form part of legal and regulatory frameworks with which a PYRAMID-compliant Exploiting Platform will need to comply, e.g. a Crash Survivable Memory Unit (CSMU, or ‘Black Box’). Whilst data retained for such a function is likely to have strict requirements such as accuracy or the inability to delete data, or the use of specific secure and survivable storage media, the fundamental process of a component identifying and capturing data items remains unchanged.

The specific recording requirements needed to comply with a given legal and regulatory framework will be an Exploiting Programme-specific issue, as it depends on platform type and capability and the countries and regions where the platform will be operated.

A.2.12.3 Overview

This PYRAMID concept sets out the approach to retaining data within a PYRAMID deployment and can be summarised as:

- Each component will have its own retention strategy for the data it handles, covering what is retained and for how long.
- Retention of specific types of data may be required for regulatory and audit purposes.
- What is retained and for what purpose will be specific to the Exploiting Programme.

A.2.12.4 Component Composition

This PYRAMID concept identifies a number of generic component behaviours in relation to Recording and Logging. These behaviours are formally captured as component responsibilities in the PYRAMID Technical Standard, section Component Composition, along with associated component services. These responsibilities and services are also summarised in Table 13: Recording and Logging PYRAMID Concept Summary.

A.2.12.5 Retention Strategy

A.2.12.5.1 Component Retention Strategies

Each Exploiting Programme will have its own requirements for retaining data created through the operation of its systems. Some of this retained data is required in order to perform its system functions (e.g. a value to be held for a short period to enable computations to be performed) whilst other data will be required for much longer, to support post-mission evaluations and audits, etc. This data will be produced by the individual components and each component will be responsible for the retention of all data deemed necessary. This is identified and managed by the component's retention strategy.

The retention strategy needs to be understood ‘within’ a component, to allow a component to identify data items and their associated context and hence respond to an outside instruction about that data. Note: translation of retained data for use by external systems (e.g. for mission debriefing or analysing health data) is outside the scope of this PYRAMID concept.

In some cases, there is likely to be alignment between the retention strategies of different components in order to capture related data from multiple components associated with a specific concept or operation. In order to facilitate alignment, the processes that result in retained data are coordinated across the system. For example, recording activities may need to be coordinated in their timings and ranges of data to be recorded/logged, where a common trigger might initiate activities across multiple components which need to record to an appropriate level of detail.

Ideally the retention strategy for a component should be a data-drivable set of rules; allowing rules to be changed between (or potentially during) operations.

A.2.12.5.1.1 Retention Strategy Requirements

The retention strategy consists of a set of specific rules and supporting information about what data the component is required to retain. The strategy also includes how the component should manage the active rules and under what conditions changes to the currently active rules are made.

Retention rules will cover:

- How the component should work out which items need to be retained for subsequent publishing or storage.
- How the data is to be sampled (e.g. the frequency of recording the value of a given parameter and the accuracy to which it is to be recorded).
- The conditions under which the component is to start or cease retaining a piece of data (e.g. to start or stop recording the value of a given parameter). The circumstances permitting this instruction may be an event internal or external to the component itself; an example of this is a rule that commences the logging of a measured parameter that is not recorded continuously but only when it falls below a defined value understood by the component, or an external instruction that recording is no longer required.
- How long a data item of a specific type (e.g. the value of a parameter at a specific instant) is required to be retained for (and hence when it is no longer required and can be disposed of). As with starting or ceasing retaining data items, the authority to dispose of a specific data item may be generated purely internally to the component or may depend on external confirmation that the data item is no longer required by another component or external user. Physical disposal of retained data in response to a change in the active retention strategy is described in the Storage PYRAMID concept.
- The classification associated with storing or transmitting the data item, which could increase the need for encryption.

The retention strategy will need to respect the requirements for mission analysis and safety and security logging, meaning that logged data will not be disposed of if there is a requirement for its ongoing retention and that some pieces of data may be retained that are not normally required by the component.

A.2.12.5.1.2 Retention Strategy Exclusions

A component's retention strategy will not include:

- Which other components or external entities require retained data items, as this is the responsibility of the Information Brokerage component.
- Decision-making covering sanitisation of data. This is covered by the Storage PYRAMID concept.

A.2.12.5.2 Outline Example

Figure 84: Example Use of a Retention Strategy shows an outline example of a component's retention strategy in use. The currently active rules in the component's retention strategy start and stop

recording a specific data item in response to an instruction from another component or an external entity.

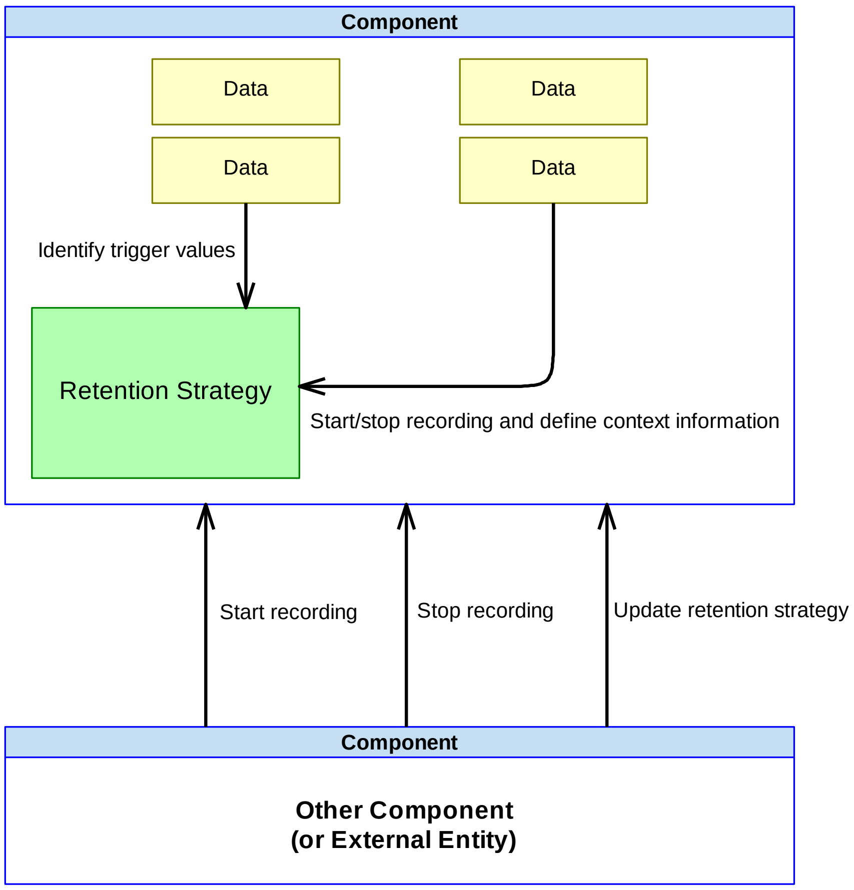


Figure 84: Example Use of a Retention Strategy

Subsequently, a change to the currently active rules is made such that an additional set of rules become active. Under the new retention rules, in addition to recording the data item in response to external instruction, the component will record the data item whenever a different parameter within the component exceeds a specified triggering value.

Note that this is only one example of how a retention strategy may operate. A component can have a retention strategy that requires no input from external entities.

A.2.12.6 Security Logging

In addition to information retained for operational reasons, including any audit purposes, it is expected that each Exploiting Programme will have a requirement to log specific events that arise during its use in order to support the security objectives of integrity, accountability, auditability, non-repudiation and to facilitate forensic examination of cyber incidents, etc. These attributes are described in the Security Approach PYRAMID concept. The data being logged will typically include:

- Anomalous behaviour or states.
- Tamper events.
- Authentication successes and failures.

- Authorisation successes and failures.
- Application and system requests to non-whitelisted resources.
- Application and system start-up, shut-down or pauses.
- Use of higher-risk functionality (certain network connections, changes to user or application privileges, use or change of cryptographic material, etc.).
- Changes to high-value data (including classification).
- Changes to relevant protocols.
- Changes to configuration files.
- Excessive use of certain resources.
- Any other cyber-related data.

The attributes being recorded for each event type are again the responsibility of the Exploiting Programme to define, but as a minimum should detail the who, what, where and when for each event.

A.2.12.7 Recording and Logging Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Recording and Logging Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept identifies the following generic component behaviours in relation to recording and logging: • Individual components are responsible for understanding the data retention strategy in relation to their own subject matter (recording) and associated processing (logging). • Individual components are responsible for identifying, recording and logging data in accordance with that strategy. • Individual components are also responsible for coordinating their recording and logging activities in accordance with their retention strategy (e.g. to coordinate timing, or to enable the collation of related information). Coordination may be achieved through alignment of component retention strategies or direct component interaction.	The PRA defines the following generic component responsibilities and services in relation to recording and logging: **Responsibilities:** • determine_retention_requirements • manage_data_retention_and_storage • coordinate_retention_activities **Services:** • Retention_Requirement • Retention_Coordination_Dependency
Specific PRA Component support for the PYRAMID Concept	The PYRAMID concept does not identify any specialised subject matters in relation to recording and logging.	N/A
Exploiter considerations	The following recording and logging related considerations are a matter for individual Exploiting Programmes: • What information needs to be retained and for what purpose. • The specific recording requirements needed to comply with a given legal and regulatory framework.	The PRA does not mandate a specific approach in respect of these considerations.

Table 13: Recording and Logging PYRAMID Concept Summary

A.2.13 Cyber Defence

A.2.13.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Health Management
- Security Approach

A.2.13.2 Introduction and Scope

This PYRAMID concept explains how the PRA can be used to provide the system with a level of security monitoring and protection from unauthorised interactions. It describes how components should work together to protect against different types of cyber attack.

A.2.13.2.1 What is Cyber Defence?

Cyber defence is the action of defending against or resisting a cyber attack. The security risks applicable to an Exploiting Platform will need to be analysed by the Exploiting Programme, but may include loss of confidentiality, integrity or availability of systems and/or data through the spoofing of external resources (e.g. ATS or GNSS), infiltration of the ground facilities by hostile forces, intercepting or replicating communications, access to sensitive data in the event of a crash, etc.

A.2.13.2.2 Why is Cyber Defence Important?

The ability to detect, defend against and recover from cyber attacks is crucial for ensuring the security and continued operability of an Exploiting Platform.

A.2.13.2.3 Technical Defences in a PYRAMID Deployment

The defences that may be required to be supported by a PYRAMID deployment include:

Protecting the boundary

- Validity and integrity checking.
- Application authentication.
- User authentication.
- Identification of spoofing.
- Coordinating infrastructure defences including networks and encryption.

Limiting disruption

- Re-routing traffic following denial of service attacks.
- Priority service for safety related data.
- Restricting access and permissions.

Limiting impact of attacks

- Segregation of domains.
- Integrity checking internal data.
- Monitoring system behaviour.
- Anomaly detection and reporting.

Detecting and analysing attacks

- Logging.
- Audit reports.
- Flagging anomalous behaviour.

The Security Approach PYRAMID concept includes information about security functions and different domains that enable the above defences.

A.2.13.3 Overview

The PYRAMID concept shows how the components contribute to the cyber defence of an Exploiting Platform by:

- Implementing protective measures that make a cyber attack more difficult to undertake.
- Detecting anomalies (unexpected states or behaviour) that may indicate the presence of a cyber attack.
- Reacting to an identified cyber attack by undertaking an appropriate contingency action.

A.2.13.4 Cyber Attack Vectors and Detection

It is not possible to address all possible cyber attack vectors within this PYRAMID concept, however, some aspects of a proactive approach to defence against cyber attack are covered in the following sections. The Defence Against Cyber Attack IV provides an example of how the PRA enables a reactive response to the effects of an established cyber attack.

The proactive PRA approach to cyber-resilience calls upon a number of components working together to monitor and control the security border of the system. This includes controlling who (authorised operators) can access the system, and with what role and level of privileges, which external entities are trusted to share information on the internal networks used by the system, detecting any threats against the system hardware, software and data, and responding to any attacks that compromise the service.

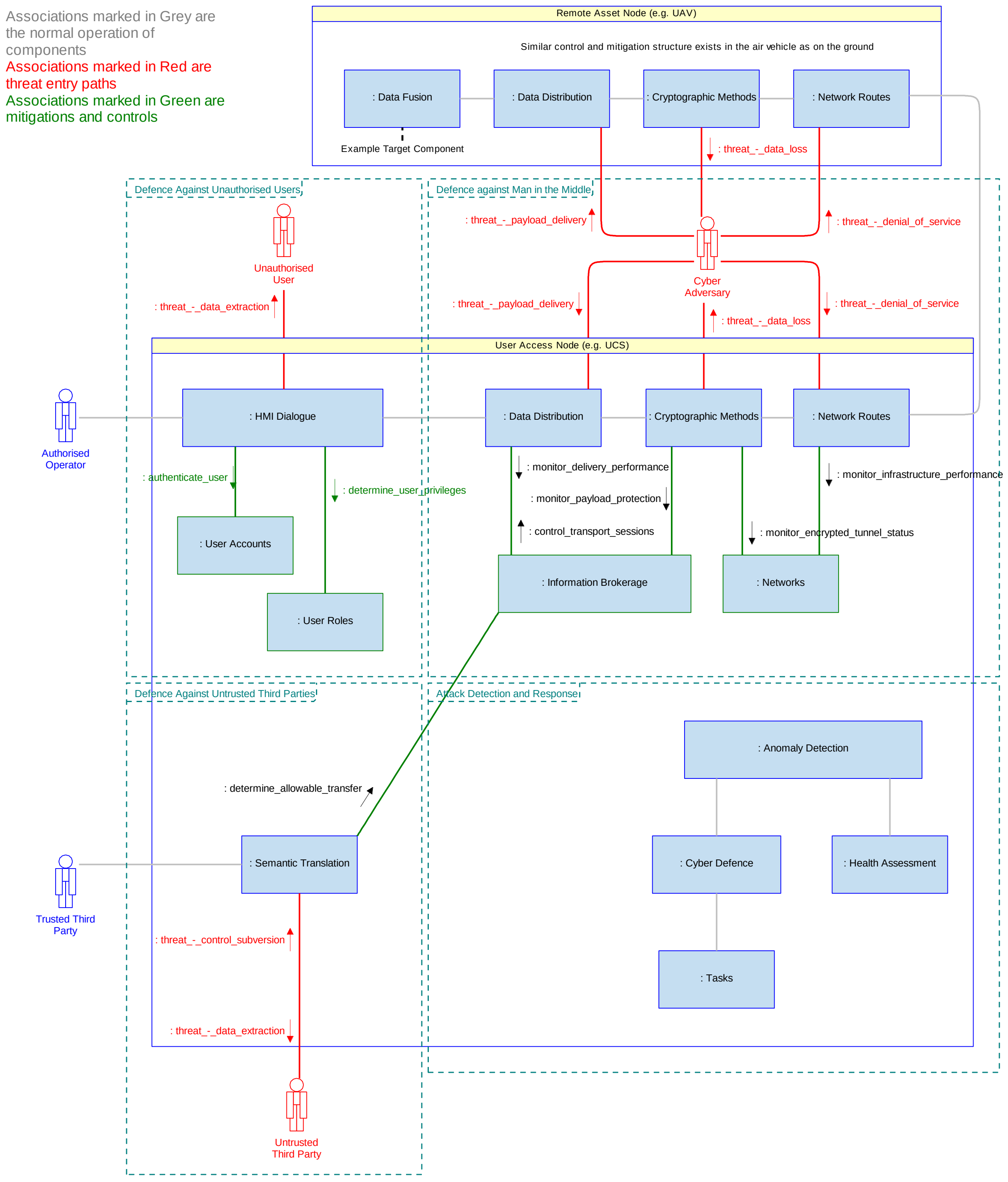

Associations marked in Grey are the normal operation of components
Associations marked in Red are threat entry paths
Associations marked in Green are mitigations and controls
Remote Asset Node (e.g. UAV)
Similar control and mitigation structure exists in the air vehicle as on the ground
: Data Fusion
: Data Distribution
: Cryptographic Methods
: Network Routes
Example Target Component
: threat_-_data_loss
Defence Against Unauthorised Users
Defence against Man in the Middle
: threat_-_payload_delivery
: threat_-_denial_of_service
Unauthorised User
Cyber Adversary
: threat_-_data_extraction
: threat_-_payload_delivery
: threat_-_denial_of_service
: threat_-_data_loss
User Access Node (e.g. UCS)
Authorised Operator
: HMI Dialogue
: Data Distribution
: Cryptographic Methods
: Network Routes
: authenticate_user
: determine_user_privileges
: monitor_delivery_performance
: monitor_payload_protection
: control_transport_sessions
: monitor_infrastructure_performance
: monitor_encrypted_tunnel_status
: User Accounts
: User Roles
: Information Brokerage
: Networks
Defence Against Untrusted Third Parties
Attack Detection and Response
: Anomaly Detection
: determine_allowable_transfer
Trusted Third Party
: Semantic Translation
: Cyber Defence
: Health Assessment
: threat_-_control_subversion
: Tasks
: threat_-_data_extraction
Untrusted Third Party

Figure 85: Defence against Cyber in a UAS Example

Figure 85: Defence against Cyber in a UAS Example represents a number of different associations between actors and components:

- Communication flows are illustrated between components that might be present in an Uncrewed Air System (UAS), comprising an Uncrewed Air Vehicle (UAV) and a UAV Control System (UCS). These are not true interactions as seen in the IVs, but represent (via the grey lines) how data may normally be moved around the UAS infrastructure, in this case between an authorised operator and the Data Fusion component (this could be for the display of track information on a screen).

- Mitigations and controls that can be mapped to responsibilities of a component are coloured green.
  More detail on interactions with HMI components is provided in the Human-Machine Interface PYRAMID concept and User Management IV, and detail on the interactions between communications components is in the Use of Communications PYRAMID concept and communications-related IVs.
- Attack vectors for a small number of threat types are coloured red. It is not realistic to illustrate all possible threat actions, so examples of possible threats from external cyber-adversaries are provided, including denial of service (DoS), control subversion, payload delivery and attempted unauthorised data access. These are prefixed with "threat" in Figure 85: Defence against Cyber in a UAS Example to further differentiate them from system actions. The main focus of Figure 85: Defence against Cyber in a UAS Example is to illustrate example entry paths available to a cyber-adversary, and where such entry paths can be proactively defended by the PRA.

A.2.13.5 Establishing Trust

Building trust in the system (the confidence that a component or other system element will behave as expected) goes further than what is possible with only the PYRAMID components and will be an Exploiting Programme decision. Typically, this might be achieved through drawing on anti-tamper mechanisms in the infrastructure and software certificates, etc. to ensure that only valid software is running on legitimate equipment. During boot up and at intervals during runtime, the system should authenticate the operation of the system and confirm that trust can be and is still established from the software down to the trust anchors in the hardware (an established authoritative entity for which trust is assumed, not derived). Once established, this baseline trusted state will form the basis by which ongoing cyber defence is performed, with suspicious behaviour such as unauthorised resource usage being analysed by the components shown in the lower right frame of Figure 85: Defence against Cyber in a UAS Example. See Attack Detection and Response for further details.

To minimise attacks through the supply chain, software whose pedigree cannot be guaranteed is expected to be sandboxed to protect the rest of the system.

A.2.13.6 Defence against Unauthorised Users

A common attack vector is for the cyber attacker to pose as a user of the system, or for a user to gain unauthorised access to additional resources or information through escalation of privileges.

The User Accounts and User Roles components would provide a first layer of boundary defence and include security enforcing functions (SEF) to limit the threat imposed by a cyber adversary attempting to gain direct access to the system via an HMI front end, authenticating the user and determining their privileges for the system and therefore protecting against the threats of control subversion, data extraction and payload delivery.

Typically, a user will be required to enter valid credentials associated with their account. It should not be possible to circumvent the login procedure.

The assignment of roles and privileges would prevent a user (including a compromised user) from elevating their permissions in order to access services and/or information to which they are not normally allowed access. It is anticipated that there will be at least three levels of user access:

- **Operator user -** the authorised operator will have access to different information or functionality in accordance with their operator and command and control roles, e.g. commander, sensor operator or maintainer.
- **Admin user -** responsible for allocating permissions to the users of the system.
- **Root user -** provides access to the operating system, application code and data within the system. This level of access will not be available to admin users or operator users, or to application software.

Actual implementation of user access and controls will be dependent on the Exploiting Programme.

A.2.13.7 Defence against Network Attacks

A.2.13.7.1 Defence against Man in the Middle

In a man in the middle (MITM) attack, a cyber-adversary impersonates each endpoint of a communication link, with the purpose of affecting confidentiality, integrity or availability of the data by intercepting and interacting with it en route.

Where the cyber-adversary seeks access to sensitive data being passed from an air vehicle to the ground station via an RF link thus impacting confidentiality or integrity, the Cryptographic Methods component would make this attack vector extremely difficult to enact assuming high grade encryption and authentication methods are utilised, thus ensuring confidentiality of information and preventing snooping on data transfers. Data hashing provides a means of identifying unauthorised modification. Commercial grade encryption keys and tokens may be appropriate for some data transfers, however some data is broadcast unencrypted.

A low tier MITM attack is the replay attack, whereby a previously valid session password is used in an attempt to impersonate a valid user (operator, admin or root). This can be prevented by the use of authentication protocols including "nonce" words, timestamping and unique random session keys, etc.

A.2.13.7.2 Defence against Denial of Service

A DoS attack attempts to restrict the legitimate availability of a communications channel, typically by flooding an unprotected network with spurious data requests or by disrupting connections between nodes. The Network Routes component generates traffic flow performance information over a network link, providing the Networks component with knowledge of a possible attack if the flow is not as expected, and allowing communications to be re-routed, limiting the number of available attack paths and also providing a priority service to any safety related data transfers. As described in the Health Management PYRAMID concept, the Anomaly Detection component may also support the detection of an attack using information from the Network Routes and Networks components.

A.2.13.7.3 Defence against Payload Delivery

Other attack vectors may target the Data Distribution component by including an unauthorised payload (e.g. malware or buffer overflow attacks) within the messages packaged for delivery. The communications components will be required to work together to protect the system borders, including by managing the communication session to reduce the bandwidth available to the adversary to exploit, should other defences be compromised.

Should such a payload manage to make its way into the system, any attempts to install the payload or to make use of resources that are not normally used by the software should be detected. It is expected that only valid, signed software would be permitted to execute; deviations from a valid software configuration would be reported as anomalous behaviour and any software calling on unexpected resources or data would be identified. This is dependent on an understanding of expected “good” system behaviour and established whitelists, etc. See Attack Detection and Response for further details.

A.2.13.8 Defence against Untrusted Third Parties

Further to the information transfer risks identified in Defence against Man in the Middle, information exchange with systems outside the administrative control of the current PYRAMID Deployment poses some additional risks. These additional risks include unauthorised data extraction and control subversion.

The Semantic Translation component manages the semantics of information exchange with an external system (e.g. via Link 16). It manages which external services can interface with the system, thus addressing the threat of unauthorised data extraction and control subversion. It does not, however, control the related devices (e.g. firewalls or gateways). Information Brokerage has an understanding of who the external participants in the exchange are. An example might be for the Semantic Translation and Information Brokerage components to prevent a cyber-adversary accessing tactical information through a valid connection made for the purpose of air traffic control data transfer, such as CPDLC (Controller - Pilot Datalink Communications). The system architecture can be expected to utilise segregated security domains to further protect data confidentiality.

A.2.13.9 Attack Detection and Response

A key indicator of a cyber attack is the system not behaving as expected. Events from all components are collected and analysed within the system. The Anomaly Detection, Health Assessment and Cyber Defence components work together to analyse the different event types, identify anomalies and determine if these may be the result of a cyber attack. Example activities would include:

- **System Monitoring -** identifying anomalies that are related to changes in the health of the system hardware. From a cyber defence point of view these may lead to additional vulnerabilities not normally present (like the loss of redundant hardware, making an attack against system availability easier).
- **Performance Monitoring -** identifying anomalies that are related to whether services are operating within expected behaviour profiles. From a cyber defence point of view a significant change in capacity use (e.g. a lot of processor time being used) could indicate a cyber attack.

- **Protective Monitoring -** identifying suspicious behaviour that can be related to security related activities, for example, an escalation of privileges by a user (e.g. to admin or root), combined with unexpected access to storage data could indicate an attack against the confidentiality of data held in the system.

When a cyber attack is identified, it will be the responsibility of the Cyber Defence component and, where possible, the attacked component to mitigate the effects and to report it appropriately so that the effects of the cyber attack may be addressed, such as disregarding a source of information or sanitising data. The Defence Against Cyber Attack IV provides an example of how the PRA responds to the effects of an identified cyber attack.

A.2.13.10 Cyber Defence Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Cyber Defence Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PRA defines a set of application layer components that may contain security related functions (SRFs) and security enforcing functions (SEFs) in support of the protection against, detection of and response to a cyber attack. Where the need for SRFs and SEFs is identified, these should be implemented in accordance with the PRA subject matter definition. The PYRAMID concept does not identify any generic component behaviours related directly to cyber defence. However, In principle, any component may support the defence against a cyber attack as follows: • By providing one or more SRFs or SEFs, so long as the function falls within the scope of the components subject matter. • By providing subject matter information that may be used in the identification of a cyber attack. • By responding to requests to take action, or constrain actions, that fall within its subject matter.	The PRA does not define any generic component responsibilities or services related directly to cyber defence, but does define specialised component services that may be used to provide information, or take or constrain action, to support defence against a cyber attack. The specialised services are defined on a per PRA component basis. In principle, any PRA component can provide a SRF or SEF, so long as the function falls within the scope of the PRA components subject matter. The PRA does not mandate the implementation of any such functions, but each PRA component definition identifies the nature of SRFs and SEFs that the component could potentially include given its subject matter. The PRA assigns indicative security classifications to components, based on an assessment of generic security risk within the context of a military security environment. These indicative classifications are not requirements on an Exploiting Programme.
Specific PRA Component support for the PYRAMID Concept	The PRA separates the following areas of functionality that relate to cyber defence: • The detection of system anomalies that might indicate a security breach (or a system health issue). • The rules and conditions that govern the authorisation of actions. • The management and distribution of cryptographic material. • The process of applying cryptographic transformations. • The identification of, and response to, cyber attacks. • Control of user access including login and authentication. • Control of user privileges.	The PRA defines the following components whose primary role relates to cyber defence: • Anomaly Detection • Authorisation • Cryptographic Materials • Cryptographic Methods • Cyber Defence • User Accounts • User Roles A range of additional PRA components play a significant role in cyber defence, as indicated by the SRFs and SEFs identified within the component definition, e.g. Health Assessment, Information Brokerage and Semantic Translation.
Exploiter considerations	The following cyber defence related considerations are a matter for individual Exploiting Programmes: • Specifying the security environment and security risks applicable to the Exploiting Programme. • Defining the security architecture including: • How components and information should be classified. • The specific approach to system partitioning to meet security requirements, for example, the use of component variants and instances (e.g. within multiple security domains), and the use of extensions. • Demonstrating that the security targets have been met.	The PRA does not mandate a specific approach in respect of these considerations.

Table 14: Cyber Defence PYRAMID Concept Summary

A.2.14 Human-Machine Interface

A.2.14.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

- Interaction with Equipment

Read in conjunction with

- Autonomy
- Component Connections

A.2.14.2 Introduction and Scope

This PYRAMID concept explains how components can be used together to support interaction between human users and system (i.e. machine) elements within an Exploiting Platform.

A.2.14.2.1 What is an HMI?

A Human-Machine Interface (HMI) connects humans to a system within an Exploiting Platform, providing users with the ability to control and monitor the system. Its main purpose is to perform translation of:

- Actions performed by users into information and instructions to be enacted by the system.
- Data held in the system into information presented to users.

These translations will occur during HMI exchanges between the system and users, which can usually be divided into a series of constituent HMI interactions. HMI interactions involve bi-directional provision of information, as described below.

In the user input direction, examples include:

- A linear button press (physical or virtual), or lever movement.
- A more dynamically defined user input, such as a voice command spoken into a microphone, a shape drawn on a surface, a hand signal or eye movement in view of a camera, or data entered via keyboard.
- Interaction with the relevant device for iris, fingerprint or speech recognition.

In the system output direction, examples include:

- Immediate illumination of a button to indicate a press was registered.
- An on-screen notification to indicate completion of the ultimate system action requested via a button press.
- A periodic sound or haptic vibration to indicate that a piece of equipment is in operation.
- More complex, integrated outputs, ranging from a dynamic map or weather forecast (which can be interactive via user input) to an augmented reality presentation such as a helmet with its own internal screen, 3D audio, voice input and touch feedback gloves.

The above examples are intended as initial, illustrative examples, and are not indicative of the full scope of potential input and output methods.

The system’s fundamental concerns, with regards to its HMI, are:

- The information that needs to be exchanged across the HMI so that it can be understood by the consumer.
- How the information should be conveyed so that it can be understood by the consumer.

A consumer in an individual interaction is either a system element receiving information instigated by a user input, or a user receiving information from the system. An exchange may consist of multiple interactions, meaning that both a system element and a user can act as both an information source and a consumer within one exchange.

A.2.14.2.2 Why is HMI Important?

The HMI is an important part of the PRA because without it users cannot interact directly with the system. By providing the connection between information that is understandable to the system and information understandable to users, HMI enables human oversight and control of the system. This separation between the user and the system by an HMI can support safety and security partitioning.

A.2.14.2.3 Exclusions to this PYRAMID Concept

The HMI supports authorisation, and while it should be considered with respect to user input and security controls, the authorisation process is excluded from further discussion in this PYRAMID concept. The User Management IV specifically covers the topic of user access to the system. The Autonomy PYRAMID concept includes a section on authorisation.

A.2.14.3 Overview

The HMI is the interface between the user and the system, and may provide information to, or receive information from a user, in any format that can be sensed and understood (for example light, images, sounds, movement or other means).

The HMI Dialogue and Information Presentation components manage the format and information flows between a user and a system such that system oversight and control can be achieved. These components provide a modular approach to the coordination of information by an HMI and decouple the system from the user interfacing mechanisms. Thus facilitating an HMI implementation in a manner that best meets an Exploiting Platform's requirements.

A.2.14.4 Component Composition

This PYRAMID concept identifies a number of generic component behaviours in relation to Human-Machine Interface . These behaviours are formally captured as component responsibilities in the PYRAMID Technical Standard, section Component Composition. These responsibilities are also summarised in Table 15: Human-Machine Interface PYRAMID Concept Summary.

A.2.14.5 Use of HMI

A.2.14.5.1 Interface between System and User Understanding

The HMI provides a connection between system and user understanding, and establishes the required dialogue that collates appropriate context so that information is comprehensible to each consumer. This prevents functional pollution, and enables systemic behaviour to be reused across multiple Exploiting Programmes with minimal change, even where the programmes have completely different HMI methodologies.

The conversion between system and user understanding is demonstrated in the following example:

- An authorised operator updates an altimeter reference pressure setting using a continuous dial input.
- An appropriate component, such as Environment Integration, will have knowledge of what an altimeter reference pressure setting means within its subject matter, and will know how to use such information (for example as a datum pressure) when it is provided via the HMI.
- The HMI implementation, meanwhile, will understand how input is conveyed via the continuous dial, and can translate dial interactions into requests for the appropriate component to change the reference pressure value. The behaviour as a result of these user inputs is the concern of the enacting component(s).

A.2.14.5.2 Human Oversight and Control

Although many functions of the system may be automated, some system activities will require some degree of human interaction to ensure safe and effective operation. The HMI will need to be able to support system operation at varying levels of autonomy, requiring the presentation of different levels of information content and degree of interaction (see the Autonomy PYRAMID concept). For example, human oversight and control may be mandated by operational regulations (e.g. airspace access) and rules of engagement. Likewise, maintenance activities will also generally require interaction with at least one human maintainer, especially in response to unexpected situations.

A.2.14.5.3 Safety and Security Partitioning

The HMI needs to provide a consistent and usable user interface whilst maintaining robust partitioning between different functional integrity levels and security domains (as described further in the Partitioning section). The HMI can also contribute to user controls, to ensure that a user can only make inputs and can only access information appropriate to their role or security clearance.

A.2.14.6 HMI Components

The aim of the HMI can be decomposed into six general functions:

- Providing the boundary connection between the system and its users.
- Organising the information needed for meaningful exchange between the system and users.
- Providing consistent HMI across the system, where applicable.

- Conveying the presentation of information in exchanges between the system and users. This includes the generation of subject specific presentations, and the consumption of inputs presented by users.
- Merging of subject specific presentations to produce an integrated output.
- Understanding system infrastructure and the hardware utilised for user interaction so that received and provided information is appropriately processed.

To meet these aims, the PRA includes two components that are specific to the HMI:

- HMI Dialogue - Source data needs to be comprehensible by its consumer. This component enables such comprehension by managing dialogue between HMI interaction participants.
- Information Presentation - This component is primarily concerned with how information is conveyed and perceived in an HMI interaction.

The PRA HMI philosophy is to provide components that can be deployed in different ways to produce interfaces that meet the needs of specific operator roles during particular mission phases, considering the operational environment and system configuration. The HMI components support this philosophy by decoupling the behaviours of HMI dialogue participants from the user interfacing mechanisms, such that they can change or be developed independently of each other in a compatible manner. This will allow Exploiting Programmes to change the HMI with as little impact on the dialogue participants as possible. HMI considerations are contained within the HMI components and not distributed through the system, so changes in other components don’t require the HMI components to change, and vice versa.

A PRA component may act as either a data source or information consumer in an HMI interaction: as the origin of system data for presentation, or as a consumer of requests to initiate some form of system behaviour. Similarly, a user and the HMI devices that they utilise may provide source input data or consume presented information. The dialogue between participants enables the system to establish comprehension rules so that a connection can be formed between system and user understanding. The context of a dialogue will inform the comprehension rules, allowing the HMI to be implemented in a flexible and versatile way.

An HMI exchange does not strictly require both HMI Dialogue and Information Presentation:

- For a linear, non-dynamic exchange, HMI Dialogue may not be necessary. For example, a dedicated text display that only presents a certain variable that is constantly displayed or requested simply (such as via a button that is not linked to user permissions), will not require dialogue that needs to be managed.
- Where device equipment contains its own capability to carry out information conveyance processing, Information Presentation may not be necessary. Such devices may range from 'smart' equipment with some degree of intelligence, to simple equipment directly linked to a dedicated information source or consumer.

The HMI components have knowledge of how and when information is presented for output, and may have some logic that can filter inputs from a user in order to reduce other components performing nugatory processing.

Note that no services are defined for PRA components to provide data for use by an HMI (see PYRAMID Technical Standard, Ref. [1], section Component Services Not Defined by the PRA).

A.2.14.7 Exchange across the HMI

A.2.14.7.1 HMI Interactions

An exchange between the system and a user may consist of multiple interactions. These interactions can be categorised into groups in the exchange process, as described below.

Source data provision to HMI components:

- Provision from a user, in which user action information is provided via an HMI device as source data.
- Provision from the system, in which system information is provided as source data.

Presentation conveyance:

- Conveyance to the system, in which input information presented to a device is captured and conveyed to the system. Information Presentation determines the most appropriate manner to convey the information.
- Conveyance to a user, in which cohesive compositions of system information are conveyed to the user via presentation resources. Again, Information Presentation determines the most appropriate manner to convey the information.

Dialogue:

- HMI Dialogue creates and manages dialogue between the system and users. The dialogue determines interaction context, which is used to comprehend the source data as information appropriate for the consumer, and to determine where the information should be sent. This includes, for example, identification of the need for additional context information, or compiling source data and context information together.

To demonstrate how interactions in these groups make up an exchange, consider a scenario where a user requires details on a payload:

- **User source data provision to HMI components:** The user performs a computer mouse click on an "Item Details" graphical element. The computer mouse device then provides data that represents the user action to the HMI components. Information Presentation instances or variants are responsible for all presentation aspects of the Graphical User Interface (GUI) that the mouse is connected to.
- **Presentation conveyance to user (input confirmation):** Information Presentation may convey a confirmation that the user input has been registered, such as a sound or on-screen notification.
- **Presentation conveyance to system:** Information Presentation instances or variants convey the mouse click as a request for details corresponding to the graphical element.
- **Dialogue for system understanding:** HMI Dialogue gathers context and determines that inventory data is required, then manages comprehension rules that enable the system to identify that the element represents a specific payload.
- **System source data provision to HMI components:** Inventory provides payload item data.

- **Dialogue for user understanding:** The dialogue enables the payload item data to be interpreted so that it is understandable for the user. Other data sources may inform HMI Dialogue of supporting data to be compiled into understandable information (for example, that another user has reserved the payload for use, or that the payload should not be released due to the current mission situation).
- **Presentation conveyance to user (satisfying user intention):** Information Presentation conveys the item details in a meaningful way relevant to the circumstances. For example, it may convey the details to fit a display area by choosing a specific size; or it may convey the details by spoken word if, for instance, a user cannot view a display or has chosen an audio conveyance method.

Figure 86: Example HMI Exchange illustrates the main aspects of the above scenario, and includes additional component instantiations to demonstrate a modular HMI implementation. For example, the additional components may be utilised where a user performs an action with the intention of receiving video from a camera monitoring a payload. The video may be presented on the same Display Screen as the payload details, perhaps through its own window. However, the Context Providers actor may include providers of user role information, and that information may indicate that the user does not have permission to access the video. In such a case a Private Monitor Screen may be connected from which an authorised operator can access the video while the other user views the Display Screen.

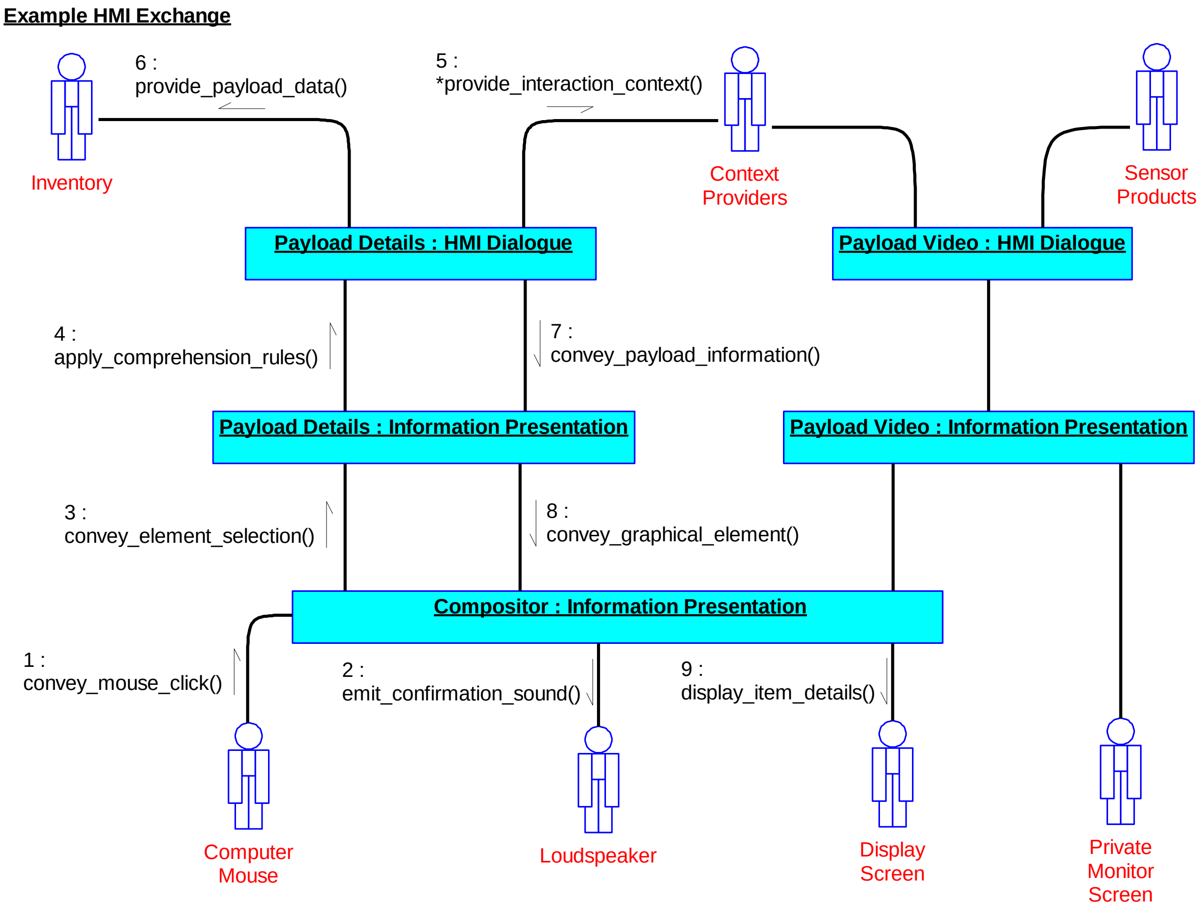

Example HMI Exchange
Inventory
6 : provide_payload_data()
5 : *provide_interaction_context()
Context Providers
Sensor Products
Payload Details : HMI Dialogue
Payload Video : HMI Dialogue
4 : apply_comprehension_rules()
7 : convey_payload_information()
Payload Details : Information Presentation
Payload Video : Information Presentation
3 : convey_element_selection()
8 : convey_graphical_element()
Compositor : Information Presentation
1 : convey_mouse_click()
2 : emit_confirmation_sound()
9 : display_item_details()
Computer Mouse
Loudspeaker
Display Screen
Private Monitor Screen

Figure 86: Example HMI Exchange

A.2.14.7.1.1 Input Validation and Filtering

The input validation and filtering performed by the HMI components includes:

- Basic validation of user inputs (e.g. the discounting of non-alphabetic keyboard characters as a result of character entry rules or filtering out background noise). Note: More complex validation, such as a check that an entered altitude setting is compliant with current air traffic services constraints, will be performed by the component that holds this information after dialogue (e.g. Environment Integration).
- Disregarding inputs for which no system component should take action (e.g. a cursor selection that is not on an interactive graphic element, or is on an element that is disabled).

Device equipment may provide an initial filter of user inputs. For example, a keyboard may eliminate erroneous keyboard presses caused by key bounce, or a microphone may reject background noise. However, in these examples, the equipment is external to the system so its filtering capability is not relevant to the HMI components.

A.2.14.7.1.2 Information Conveyance and Dialogue

Information Presentation will determine the intent of user input and system output. For a GUI this may include consideration of aspects such as window layering and transparency, whereas for an audio interface this may include aspects such as speech recognition for voice input or volume considerations for output sounds.

The following examples focus on an HMI with cursor selection and visual display capability. In Figure 87: Information Conveyance and Dialogue Example, Information Presentation determines that a cursor selection event within a visual presentation area is associated with the graphical element representing the switch controlling the undercarriage lights, and conveys this to the system. HMI Dialogue uses interaction context (such as the state of other system activities) and its predetermined rules to comprehend the input data as a request for the appropriate component, in this case Lights, to perform a setting change. The dialogue may identify further affecting context, such as information that a certain light level is required in the current phase of flight or weather conditions, which may prompt further confirmation of the user intention.

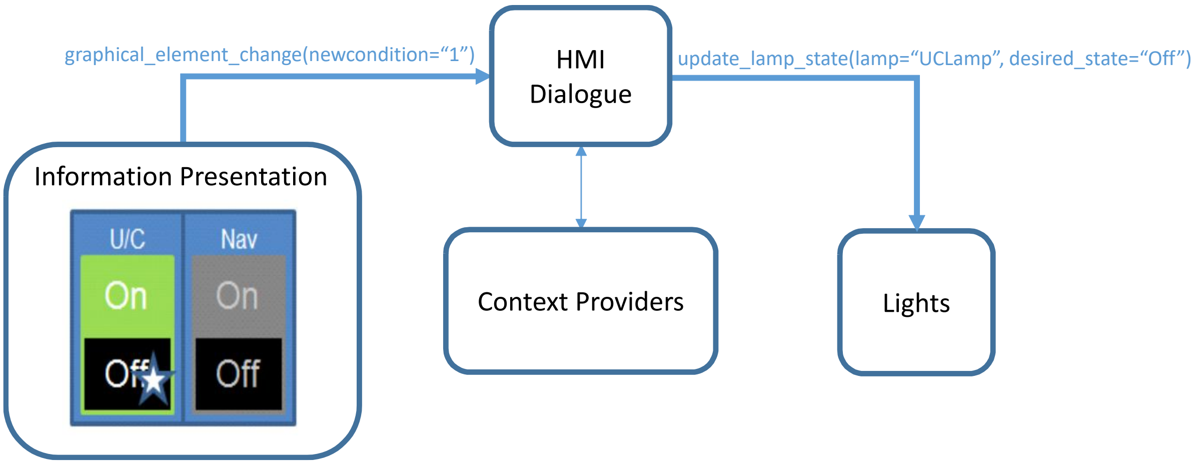

graphical_element_change(newcondition="1")
HMI Dialogue
update_lamp_state(lamp="UCLamp", desired_state="Off")
Information Presentation
U/C
Nav
On
On
Off
Off
Context Providers
Lights

Figure 87: Information Conveyance and Dialogue Example

HMI Dialogue can gather or compile data from multiple user or system sources and map it to parameters that Information Presentation characterises as elements of the HMI.

The PRA does not (and cannot) mandate patterns of behaviour for an HMI Dialogue component as they are determined by an Exploiting Programme. The translations performed by HMI Dialogue will usually be developed to suit the requirements of a specific programme.

In Figure 88: Parameter Mapping, HMI Dialogue maps data obtained from three components to parameters that Information Presentation can use, in order to display a virtual ‘switch’ output in the ‘On’ condition (because the Lights component says the lamp is on), with input enabled (because the aircraft is performing a 'landing' activity) and in green (because the lamp has no fault indications).

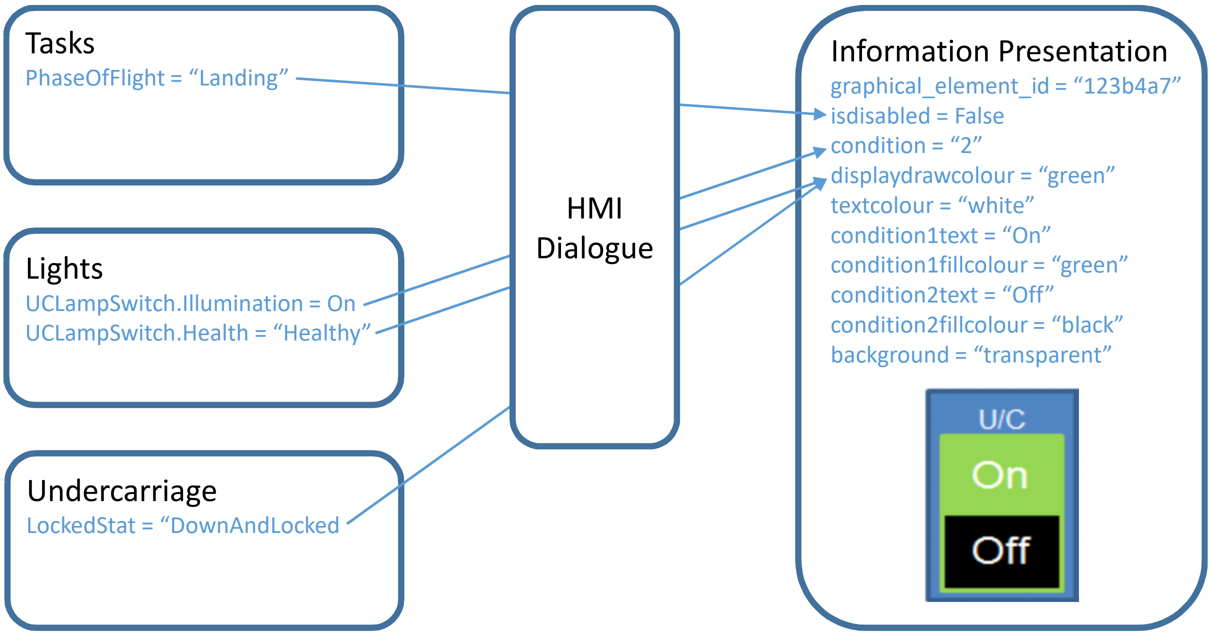

Tasks
PhaseOfFlight = “Landing”
Lights
UCLampSwitch.Illumination = On
UCLampSwitch.Health = “Healthy”
Undercarriage
LockedStat = “DownAndLocked
HMI Dialogue
Information Presentation
graphical_element_id = “123b4a7”
isdisabled = False
condition = “2”
displaydrawcolour = “green”
textcolour = “white”
condition1text = “On”
condition1fillcolour = “green”
condition2text = “Off”
condition2fillcolour = “black”
background = “transparent”
U/C
On
Off

Figure 88: Parameter Mapping

A.2.14.7.2 Presentation Composition

Information Presentation delivers compositions of presentation elements. These compositions can be at a variety of scope levels, from a merging of elements defined within the component, to a combination of specialised presentations that are provided by other Information Presentation instances or variants.

In Figure 89: Presentation Composition, Information Presentation is deployed to understand the rules for presenting two independent items (both switches). In this example Information Presentation understands the graphics commands required to draw a switch, and is provided with a number of conditions that allow it to draw the desired graphics. Settings for the colours and text strings to be displayed are supplied along with more generic higher priority information that globally changes the presentation (such as the activity being ‘disabled’ and not selectable by a user).

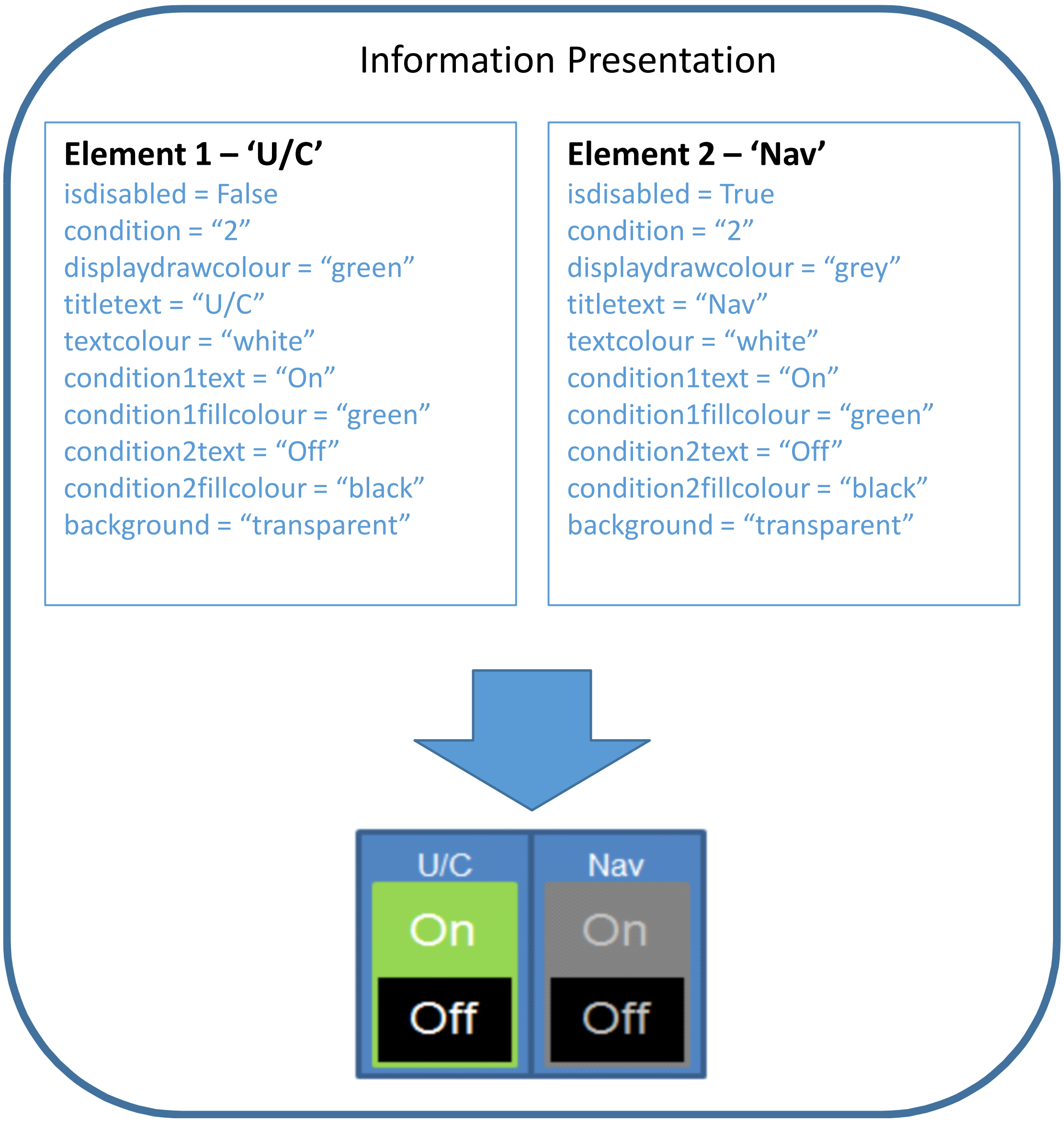


Figure 89: Presentation Composition

Figure 90: Presentation Composition of Multiple Element Types provides an example of a composition consisting of three element types that could each be fed by information from separate HMI Dialogue instances or variants. In this example, Information Presentation uses windowing to layer the information via a display. Note that compositions are not restricted to elements of the same sensory form. For instance, a series of sound alerts could be composited together with visual elements.

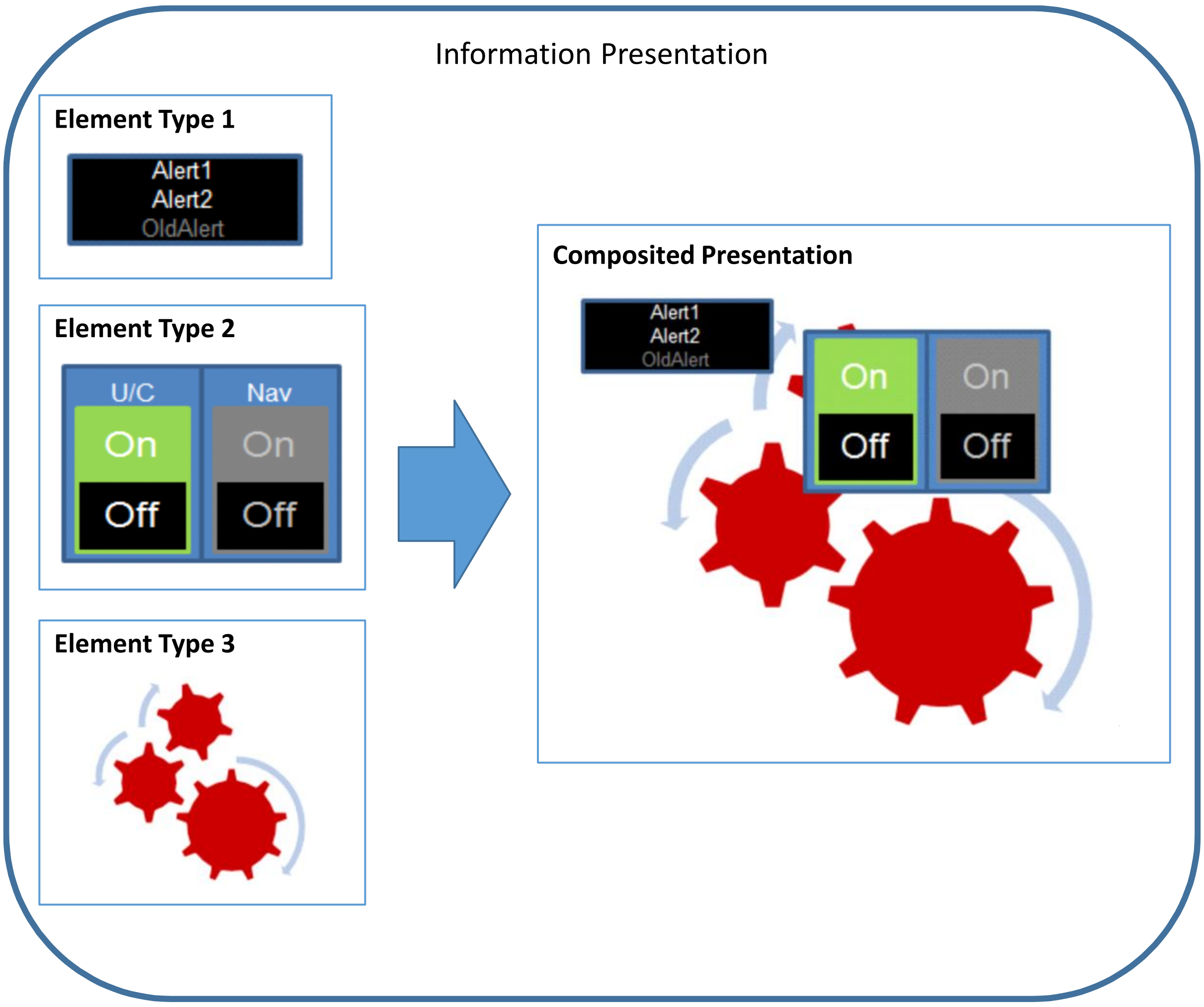


Figure 90: Presentation Composition of Multiple Element Types

A.2.14.8 Structural Considerations

This section summarises design considerations for the HMI components within a deployment. This includes decisions around modularity, partitioning and coordination of HMI-specific information.

A.2.14.8.1 HMI Modularity

The dialogue required to comprehend certain forms of source data will be more effective when the coordinating HMI Dialogue component is highly focused on the HMI exchanges that involve the specific source data types. Similarly, to optimally prepare information so it is available to specific system elements and presentable in certain forms, the dialogue will benefit from tailoring for those HMI exchange participants. This can be achieved by limiting the range of HMI exchange types that an HMI Dialogue instance will manage, and tailoring the instances for those exchanges through data driving. Such modularity and the customisation it allows will enable simpler interfaces, and redevelopment of HMI behaviour will therefore require less effort.

Modularisation of presentation compositions should also be considered, as this will support the design and modification of complex, 'system of systems' presentations. The graphical elements that make up such display compositions are suited for provision by different component instantiations. For example, the windowing systems used by most PCs are highly cohesive, interactive and modifiable, offering the user the ability to move or resize windows. Modularisation enables these presentation elements to be managed independently by Information Presentation component instances that can have focused relationships with the most suitable HMI Dialogue instances, and elements like symbology, personalised displays, or widgets can be changed or replaced easily.

Figure 91: Composition from Multiple Component Instances or Variants illustrates how the example presentation shown in Figure 90: Presentation Composition of Multiple Element Types could be provided by an Information Presentation instance or variant responsible for a windowing presentation that hosts presentation elements from multiple specialised component variants. Multiple instances or variants of HMI Dialogue could also be deployed to help make the information available to each Information Presentation instance or variant in the required form.

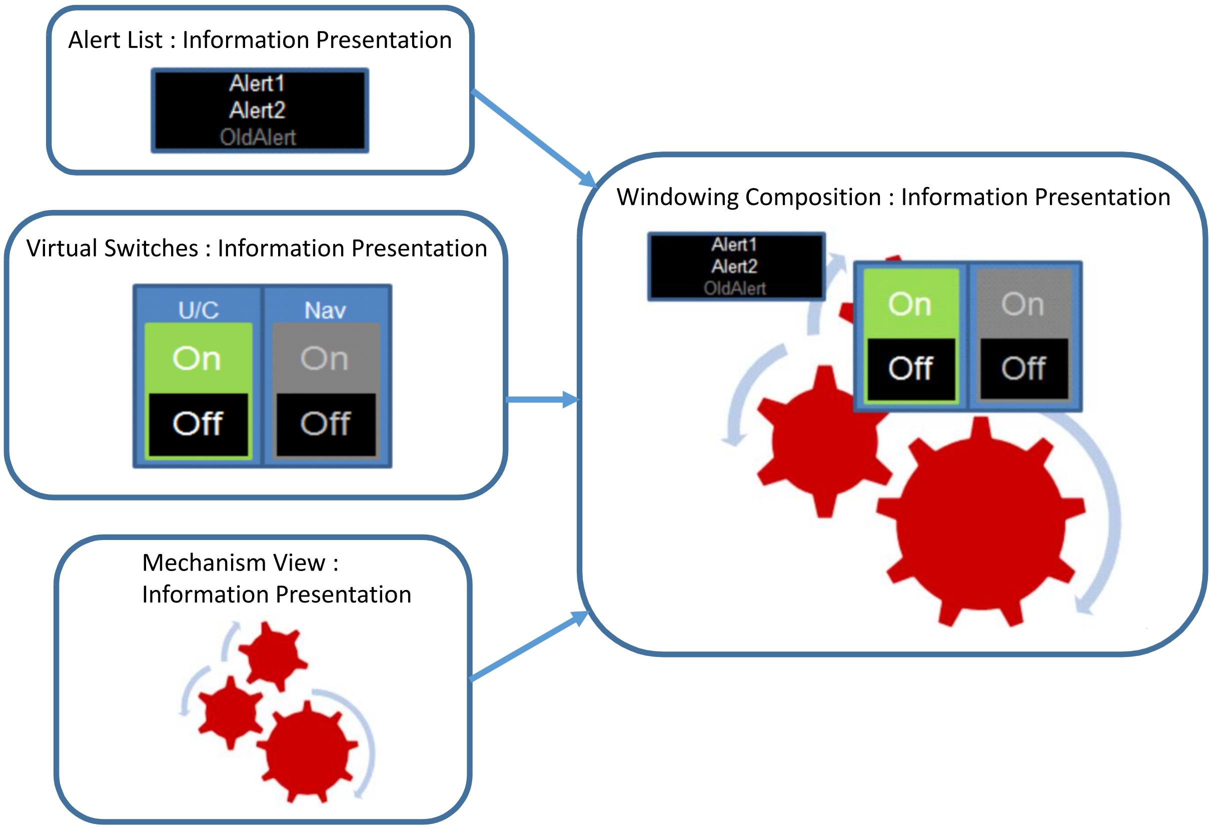

Alert List : Information Presentation
Alert1
Alert2
OldAlert
Virtual Switches : Information Presentation
U/C
Nav
On
On
Off
Off
Mechanism View : Information Presentation
Windowing Composition : Information Presentation
Alert1
Alert2
OldAlert
On
On
Off
Off

Figure 91: Composition from Multiple Component Instances or Variants

A.2.14.8.2 Devices used in HMI

Devices that users interact with can vary significantly in terms of their design, versatility and processing capability, and may be used together to provide a multifaceted interface. For example, the system may react to user input through a control stick by providing haptic feedback through a user's seat, gloves, or through the stick itself. A helmet mounted display may be used which responds to control of the stick, and the helmet may also use head or eye tracking technology. The control stick or helmet may be modifiable during or between missions, perhaps with options for different modes. The HMI components allow the use of varied device combinations through their flexibility.

The range of potential devices can be grouped by common interfacing mechanisms or specialisations. For example, a touch overlay and pressure mat may be grouped, or there may be a family of keyboard and keypad types. The relationship between devices and modular component instances or variants can allow some devices to be replaced by others from the same family with fewer required changes to the interface.

Where complex information from multiple sources needs to be presented, this drives the need for the Information Presentation component's capabilities. Such capabilities may be provided by the component itself, or by complex devices that can perform a similar role in relation to their area of concern.

A.2.14.8.3 Partitioning

The HMI components support partitioning to meet safety integrity requirements and the security objectives of confidentiality, integrity and availability.

For safety, this means that where a group of HMI components work together to handle information considered to be of high integrity (i.e. high DAL components) they can be assigned to a software partition segregated from lower DAL components. This prevents the lower DAL software from compromising the operation of the higher DAL software, and allows the cost of development to be controlled, with lower DAL software not requiring the same level of verification, etc. For example, an exchange that requires provision of highly consistent information could be conveyed more simply than, or separately from, low integrity information to avoid costly misinterpretation.

Components can also be partitioned based upon confidentiality, integrity and availability requirements with security control mechanisms enforcing separation between security domains. These controls, the nature of which is a decision for the Exploiting Programme, can impact data flow between domains and need careful consideration. For example, where displayed data is required to have high availability, the HMI data source and consumer would ideally be located in the same partition so as to prevent the control, such as a firewall, delaying or even blocking the data. Data must be handled correctly by the HMI to ensure no unauthorised data can leak across the security partition (note that the HMI is not however responsible for maintaining the security partition - see the Safety and Security PYRAMID Concepts PYRAMID concept for more detail).

Differing security clearance levels are addressed by the User Accounts and User Roles components, with the filtering and presentation of information to operators provided by the HMI components.

A.2.14.8.4 Coordination

Information will be provided by users and different system components in numerous forms, so coordination of the presented information is a key consideration for interface consistency and alignment. The only components concerned with user input and understanding are the HMI components: whether a distance is presented in user-understandable units such as metres, kilometres, data miles or nautical miles is an HMI matter, whereas the rest of the system is concerned with the value and internal representation of the distance in question.

Coordination of the forms of presented information is achieved through the design of the specific HMI in question. This coordination is maintained through bridging of HMI component entities to the associated system entities that will ultimately consume information from, or provide information to, the HMI. Consistent data driving of HMI components will also support successful coordination. The Component Connections PYRAMID concept provides further detail on this topic and related areas.

A.2.14.9 Human-Machine Interface Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Human-Machine Interface Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept does not identify any generic component behaviours directly related to Human-Machine Interface (HMI). However, all components support the exchange of subject matter information for the purpose of HMI interactions. Components are responsible for validating information received from an HMI, where the scope of validation checking is commensurate with their subject matter knowledge and reasoning.	The PRA does not define any generic component responsibilities or services, specifically for providing access to the data held internally to a component, that may be required to support user interaction via an HMI. The PRA defines the following generic component responsibility in relation data validation (regardless of the source of the data): • data_validation
Specific PRA Component support for the PYRAMID Concept	The PRA separates the following areas of functionality that relate directly to HMI: • Knowledge of the information needs of users and system elements, and the interaction rules that enable the information to be exchanged between them. • The presentation of information, in a usable and appropriate format, in order to facilitate interactions between a user and the system, enabling user understanding of system information and user input to influence system behaviour.	The PRA defines the following components whose role relates to HMI: • HMI Dialogue • Information Presentation
Exploiter considerations	The following HMI related considerations are a matter for individual Exploiting Programmes: • The specific approach to system partitioning, for example, the use of component variants and instances, and the use of extensions, in order: • To meet safety and security requirements while supporting HMI interactions. • To coordinate information across an HMI suite. • To maximise modularity and re-use of HMI components.	The PRA does not mandate a specific approach in respect of these considerations.

Table 15: Human-Machine Interface PYRAMID Concept Summary

A.2.15 Interfacing with Deployable Assets

A.2.15.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Interaction with Equipment
- Use of Communications

A.2.15.2 Introduction and Scope

This PYRAMID concept covers interfacing with deployable assets before and after deployment from a PYRAMID deployment air vehicle.

A.2.15.2.1 What is a Deployable Asset?

A deployable asset is defined as any physical hardware (e.g. role fit equipment) carried on an Exploiting Platform which can be deliberately deployed from the Exploiting Platform during a mission. Deployable assets include external assets which can be deliberately jettisoned but which in normal operation are retained aboard the Exploiting Platform (e.g. sensor pods).

Deployable assets can be of a wide variety of types, including non-PYRAMID based stores and can potentially be interfaced with in multiple ways by the PYRAMID deployment air vehicle.

A smart asset is an asset which requires a data interface with the Exploiting Platform.

A dumb asset is an asset which does not have a data interface with the Exploiting Platform, but does have utility and/or mechanical connections.

A.2.15.2.2 Why is Interfacing with Deployable Assets Important?

The ability to integrate with deployable assets, both before and after release from the Exploiting Platform, is critical to being able to carry out a multitude of mission types.

A.2.15.2.3 Exclusions to this PYRAMID Concept

This PYRAMID concept only covers the types of deployable asset and the different types of connection required between a deployable asset and an Exploiting Platform.

It is not intended to illustrate the activities required for fitting or deployment of the asset (for example as part of an attack task) or the decision making supporting deployment. Refer to the following IVs for an outline of these:

- Role Fit Discovery IV
- Plan For A/S Engagement IV
- Releasing IV
- Jettison Management IV

This PYRAMID concept does not cover the process of establishing and maintaining communications channels in detail. Refer to the following PYRAMID concepts and IVs for further information:

- Use of Communications PYRAMID concept
- Network Initialisation IV
- Connection Management IV
- Data Transfer IV
- Tactical Exchange IV
- Link Selection IV

A.2.15.3 Overview

The PYRAMID concept for Interfacing with Deployable Assets outlines the basic types of interaction between a PYRAMID deployment air vehicle and various types of deployable asset (such as a weapon or a sensor), and the components that an Exploiter would need to consider in order to implement these interaction types.

In all cases, connections with a deployable asset are managed as connections with a piece of equipment, with data connections being mapped to/from components as required on receipt/transmission (see the Interaction with Equipment PYRAMID concept for further details).

The interactions with a deployable asset will depend on whether or not the asset is PYRAMID-accessible, and what level of intelligence the asset encompasses.

A.2.15.4 Types of Connections

The following potential types of connection exist between an Exploiting Platform and deployable assets:

- **Mechanical**: This covers physical attachment of the deployable asset (e.g. clamps and lugs) in addition to providing mechanical impulse to support safe deployment. This is managed by the Release Effecting component.
- **Utilities (non-Data)**: Deployable assets, or their interfaces, may require or provide utilities, such as power (electrical or hydraulic), cooling or fuel. These are treated like any other utility. For example, a power sink is managed by the Power component.
- **Signal (non-Data)**: Discrete signals may be required for control or interrogation (for example a Mated Connection Indication or Release Consent).
- **Data**: Smart assets require transfer of data (e.g. to update plans or settings such as the impact settings of a smart fuse or the target location), or supporting built in test to report readiness prior to deployment. Data transfer mechanisms could include electrical, optical or RF and data transfers may be in either direction or unidirectional. This will be managed by the Semantic Translation, Information Brokerage, Data Distribution and Communicator components (see the Use of Communications PYRAMID concept and the Data Transfer and Tactical Exchange IVs for more information). The requirements on the interfacing protocol will depend on the physical constraints and operating environment. For example, a protocol designed for direct, close proximity operation in a safety critical airborne system (e.g. MIL-

STD-1760 Ref. [10]) would be needed for assets on-board an Exploiting Platform, whilst a protocol designed for remote, long range operation in air (e.g. TacNet) would be needed for assets after deployment.

When a deployable asset has been attached to one of the stations of an Exploiting Platform, the platform needs to know the location, type, and configuration/status of the asset. This is true for mechanical, signal and/or data type connections. This is covered further by the Role Fit Discovery IV.

A.2.15.5 Smart Deployable Assets

Smart deployable assets are those which require a data interface with the Exploiting Platform, these could be PYRAMID-based or not. Examples of this type of asset are currently most guided bombs and long-range missiles, as well as external sensor and electronic countermeasures pods.

On board an Exploiting Platform

Whilst connected to an Exploiting Platform, smart deployable assets will need a data connection that is compliant with a communications protocol supported by the asset (for example MIL-STD-1760 Ref. [10]). They will also need the ability to identify themselves to the Exploiting Platform to support role fit discovery. This connection will require the PYRAMID deployment air vehicle:

- To support translation between PYRAMID component services and the messaging format used by the deployable asset.
- To support the required transmission technologies, whether it being electrical, optical, RF or other signals in accordance with this format to communicate with the asset.
- To support the required communications protocols to route message traffic between the asset and components within the PYRAMID deployment air vehicle.

Where data is needed concerning a smart asset (such as the mass and aerodynamic properties of a guided bomb to support ballistic calculations) this is handled as data within a PYRAMID deployment in appropriate components such as Release Aiming (see the Data Driving PYRAMID concept for more information).

After deployment

After deployment, if the deployable asset supports in-flight communication (for example a Network-Enabled Weapon) it will continue to communicate with the Exploiting Platform via a protocol. Note that the protocol used to communicate via Semantic Translation and Data Distribution may change after deployment. For example, MIL-STD-1760 could be used whilst on-board the Exploiting Platform and TacNet whilst in flight. This will be similar for deployable assets which have unidirectional communications.

A.2.15.5.1 PYRAMID Accessible Deployable Assets

PYRAMID-accessible assets are a subset of smart assets built using PYRAMID components that interact directly with other parts of the PYRAMID deployment. This excludes any PYRAMID-based deployable asset which does not allow access to its components by another PYRAMID-based system. Whilst no legacy PYRAMID-accessible assets exist, it is assumed that many future external pods and stores specifically developed for a PYRAMID deployment air vehicle would themselves be designed to be PYRAMID deployments. For example, a deployable UAV carried on an Exploiting

Platform station as a reconnaissance asset, or communications or targeting relay might well be a PYRAMID deployment itself.

On board an Exploiting Platform

Whilst connected to an Exploiting Platform, a PYRAMID-based asset that is intended to be deployed to support mission goals will form part of the PYRAMID deployment air system. The components within the deployable asset can interact with those on the host platform, and can be directly involved in providing data concerning the asset to components on the platform that require it. Assets that aren't intended to be deployed but can be jettisoned if required (e.g. sensor pods) can be thought of as PYRAMID-based pieces of equipment (see the Interaction with Equipment PYRAMID concept) with the caveat that there would be some differences with regards to initial detection and disconnection.

After deployment

After deployment, if the deployable asset supports remote interaction the asset will continue to exist as a separate node of the PYRAMID deployment air system although it may use alternative interaction mechanisms.

In some cases, these connections may only be established after deployment (if some components only become active once the deployable asset has been deployed).

A.2.15.6 Dumb Assets

Dumb assets are those which do not have a data interface with the Exploiting Platform. Any interface with these assets is purely utility based, e.g. electricity or fuel. Examples of this type of asset are free-fall bombs and external tanks.

On board an Exploiting Platform

No direct data communication is possible with a dumb asset although mechanical, electrical and utility connections may exist. The Exploiting Platform may be able to sense the presence of a store on a station even if it cannot interrogate it. Where data is needed concerning a dumb asset (such as the mass and aerodynamic properties of a free-fall bomb to support ballistic calculations) these are held as data within a PYRAMID deployment in appropriate components such as Release Aiming (see the Data Driving PYRAMID concept for more information).

Any unusual behaviour exhibited by a dumb asset may still be identifiable by an instance of the Anomaly Detection component on an Exploiting Platform, and reported accordingly (see the Health Management PYRAMID concept).

After deployment

After deployment, no direct communication is possible with a dumb asset.

A.2.15.7 Nested Assets

Dumb deployable assets may be held within a larger asset which itself is smart and may be PYRAMID-based. Examples of this are countermeasure flares within a defensive aids pod, or unguided weapons within a smart carrier (such as a bomb store carrier, launcher, rocket pod or gun pod). In this case, the Exploiting Platform would communicate with the smart carrier as described for smart assets above, which would in turn interact with the asset.

It is also possible for smart assets to be deployed from smart carriers, the communications and control of which would be Exploiting Programme dependent.

Deployable assets may also be nested within smart carriers that are not in themselves deployable, but interact with the host platform as a fitted piece of equipment.

A.2.15.8 Typical Component Usage

The following examples show how a PYRAMID deployment air vehicle can interface with a deployable asset. The following statements can be made about the examples:

- The deployable asset used is the same.
- They can be used for both before and after deployment, depending on the type of interface.
- They are applicable to both PYRAMID-based and non PYRAMID-based deployable assets.
- The components used to provide information are only examples, these components will change based on the type and use of deployable asset.

Figure 92: Physically Connected Interface shows an example of a PYRAMID-based Exploiting Platform interfacing with a deployable asset through direct connections, e.g. via an umbilical cable. In this example the Exploiting Platform is preparing the deployable asset prior to deployment and providing the deployable asset with data on an object of interest.

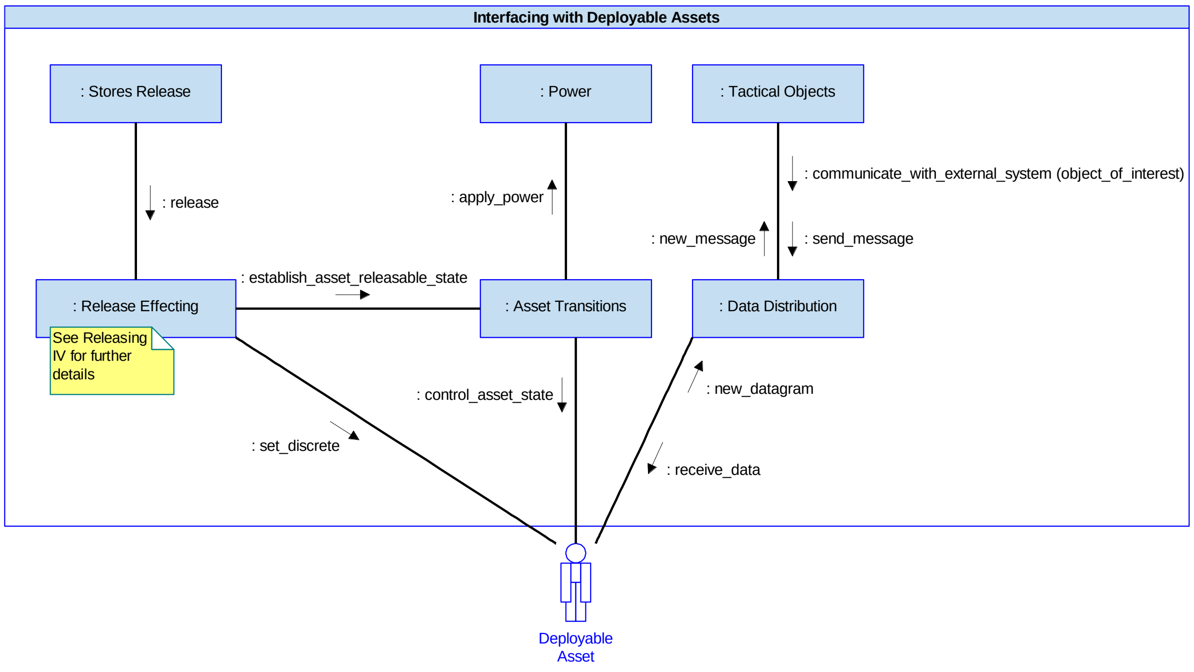


Figure 92: Physically Connected Interface

Figure 93: Remote Interface shows an example of a PYRAMID-based Exploiting Platform interfacing with a deployable asset remotely through the use of the communication based components to provide updated information on an object of interest (such as an updated location to support course correction). For remote interfaces the Semantic Translation component could utilise extensions to cater for the required data link protocol requirements, with the Data Distribution and Communicator components enabling the appropriate communication channels for the deployable asset.

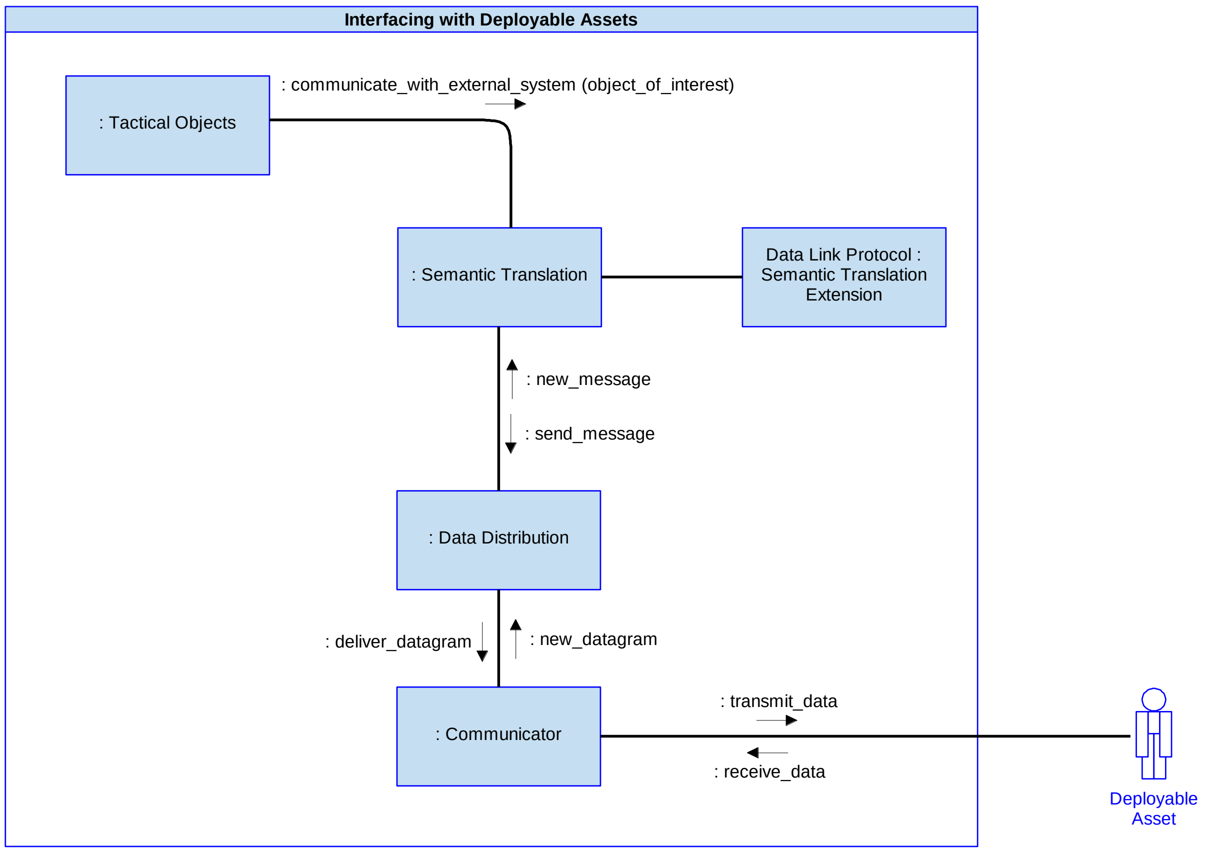


Figure 93: Remote Interface

A.2.15.9 Interfacing with Deployable Assets Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Interfacing with Deployable Assets Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept does not identify any generic component behaviours in relation to interfacing with deployable assets.	In principle, any PRA component may support interfacing with deployable assets. Connections with a deployable asset are managed as a connection with a piece of equipment, in line with the Interaction with Equipment PYRAMID concept. The PRA does not define any generic component responsibilities or services related directly to interfacing with deployable assets.
Specific PRA Component support for the PYRAMID Concept	The PRA separates the following areas of functionality that relate to interfacing with deployable assets: • The selection of a store to release and initiate their release at the appropriate time and in the appropriate sequence. • The coordination of the jettison of physical items from the platform. • The determination of a targeting solution for a given store for a specific aiming scenario. • Effect the release of a store. • Determination and verification of the systems inventory.	The PRA defines the following components whose role relates to interfacing with deployable assets: • Stores Release • Jettison • Release Aiming • Release Effecting • Inventory The PRA defines a number of components that deal with different aspects of communication that may be used for communication with a deployable asset. For more information see the Use of Communications PYRAMID concept.
Exploiter considerations	This PYRAMID concept does not identify any specific Exploiting Programme considerations.	N/A

Table 16: Interfacing with Deployable Assets PYRAMID Concept Summary

A.2.16 Tactical Information

A.2.16.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Control Architecture
- Component Extensions
- Data Driving

A.2.16.2 Introduction and Scope

This Tactical Information PYRAMID concept explains how the PRA supports the handling of tactical information and the sensor data from which it is derived.

A.2.16.2.1 What is Tactical Information?

Tactical information in the context of the PRA is defined as information about objects in the battlespace. An object in this context is an entity in the outside world which has relevance to the mission; examples include people, vehicles and buildings. This definition of tactical information also includes information about an object's relationship to the wider battlespace. These relationships are either between objects within the battlespace (such as an object threatening another), or between an object and other measureable parameters of the battlespace (such as an object reacting to a change in the electromagnetic spectrum). Tactical information is distinguished here from sensor data; tactical information is derived from sensor data but the latter represents data upon which no tactical interpretation or significance has been placed.

A.2.16.2.2 Why is Tactical Information Important?

Tactical Information is important due to it being a perception of the world that will be used to inform many decisions made by the system or an operator.

A.2.16.3 Overview

This PYRAMID concept explains how the PRA supports the management of tactical information and the handling of sensor data from which it is derived. It can be summarised with three key principles:

- The definition of tactical information components to be configurable in terms of the objects they reason about for a given Exploiting Programme.
- The separation of sensor data handling from the tactical information derived from it.
- The separation of sensor data handling from its high-level coordination and control activities.

Although this PYRAMID concept does not cover this in any detail, tactical information is not limited to data originating from the same node, it can be communicated from another vehicle/system. Tactical information can also be derived from existing mission data.

A.2.16.4 Tactical Information Components

Tactical information is split across a number of components. While they are each responsible for a particular element of tactical information, separating these elements into different components allows a variation in the complexity of the tactical understanding that different Exploiting Programmes can have. Hence any component not owning any tactical information cannot be explicitly defined as a tactical information component.

Tactical information components exist as service components in line with the Control Architecture PYRAMID concept. The set of tactical information components is as follows:

- Tactical Objects
- Threats
- Trajectory Prediction
- Data Fusion
- Geography
- Observability
- Signature
- Susceptibility
- Vehicle Performance

The key design consideration about tactical information components is that within the PRA they are defined to be configurable in terms of the objects they reason about.

Figure 94: Threat Assessment Deployment Example and Figure 95: Sensing Deployment Example illustrate two different operational concepts utilising the same component to demonstrate how driving a particular component with different data at different times (either through interactions with other components or data driving) facilitates that component's utilisation in a range of deployments.

In one case the Observability component is providing information used in the determination of threats and in the other it’s supporting the planning of a sensing task.

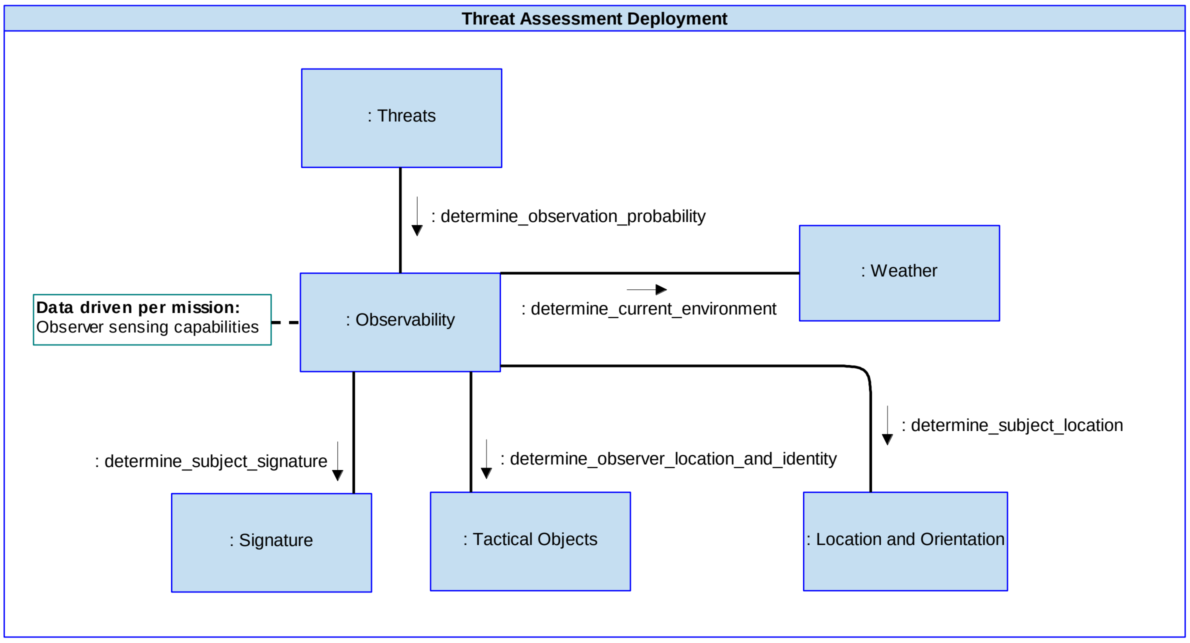

Threat Assessment Deployment
: Threats
: determine_observation_probability
: Weather
: determine_current_environment
Data driven per mission:
Observer sensing capabilities
: Observability
: determine_subject_location
: determine_subject_signature
: determine_observer_location_and_identity
: Signature
: Tactical Objects
: Location and Orientation

Figure 94: Threat Assessment Deployment Example

In Figure 94: Threat Assessment Deployment Example, Threats tasks Observability to determine the probability of being observed by an object which it has deemed to be a potential threat.

In this scenario, Observability has been configured with runtime data (see the Data Driving PYRAMID concept) that includes the sensing capability of expected observers specific to the mission. Then using this data in conjunction with information provided by the other components illustrated on Figure 94: Threat Assessment Deployment Example, the Exploiting Platform's observability can be determined and continually reported to Threats.

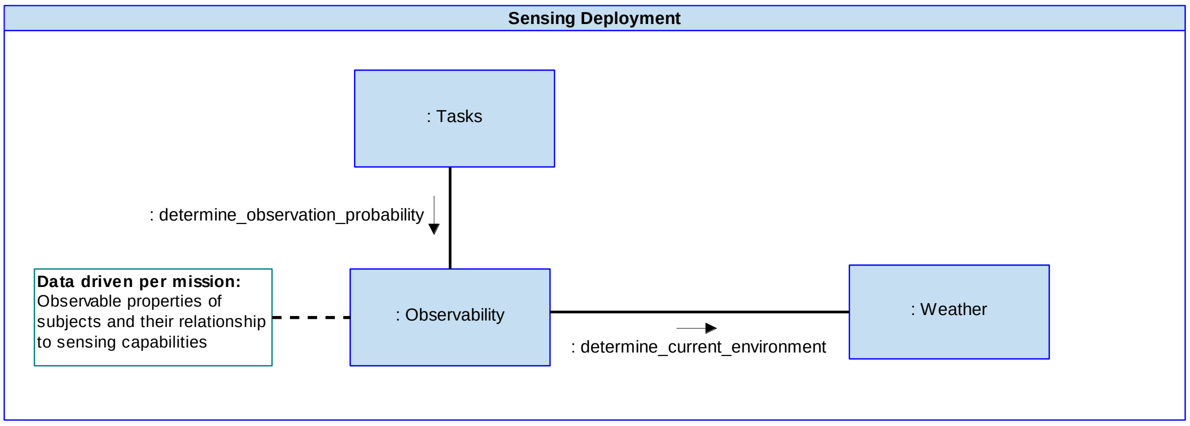

Sensing Deployment
: Tasks
: determine_observation_probability
Data driven per mission:
Observable properties of subjects and their relationship to sensing capabilities
: Observability
: determine_current_environment
: Weather

Figure 95: Sensing Deployment Example

In Figure 95: Sensing Deployment Example, Tasks requests that Observability determines the probability of observing an object type within a given region using a given capability.

In this scenario, Observability has been configured with runtime data (see the Data Driving PYRAMID concept) that includes information about the subject's susceptibility to detection (e.g. observable properties) and how this susceptibility relates to sensing capabilities. Observability would also understand the sensing capability of the Exploiting Platform either via the capability assessment of the platform or by data driving to obtain the platform's actual sensor configuration. In addition, for the purposes of this example the Tasks component is providing the required parameters to support the planning query, such as the approximate location and assumed object type of the subject, as part of its tasking to the Observability component.

Then using this information in conjunction with information determined by Weather, Observability is able to assess the probability of the object type being observed.

A.2.16.5 Separation of Tactical Information from Data Handling

Sensor data handling in Sensor Products has been separated from components that understand and reason about tactical information. Sensor Products is responsible for the manipulation and analysis of sensor data captured by sensors. Manipulation may include, for example, processing to enhance the data or the combining of data from multiple Sensors. The analysis of the data undertaken by Sensor Products involves the identification and characterisation of features or patterns within the data. Sensor Products thus generates evidence of real-world objects from sensor data, but it places no tactical interpretation or significance on that evidence. It is the role of Data Fusion to evaluate evidence of potential real-world objects and to place tactical interpretation on such information, i.e. to draw conclusions about the identity and nature of real-world objects. This process may involve the fusion of multiple pieces or types of evidence. Tactical Objects is responsible for maintaining detailed knowledge of such objects and reasons about their relationships and behaviours.

In this way, the components that reason about tactical information are separated from those that capture, manipulate and analyse the sensor data from which such tactical information is derived. This approach also separates knowledge of the traceability of sensor data used to generate evidence (such as the lineage of sensor data combined by Sensor Products) from the traceability of evidence used to generate an understanding of objects within the battlespace (such as the lineage of evidence fused by Data Fusion). Sensor Products, for example, may combine images from multiple sensors to create a multi-spectral image. Data Fusion on the other hand might combine evidence of a pattern found within such an image with evidence of an RF emitter at a similar location and time, with the interpretation that this represents a vehicle of a particular type. This is a key separation as it allows the implementation of multiple sensor data handling approaches in a given deployment while still maintaining a single source of truth for the system's view of the battlespace.

In the diagram below Data Fusion is operating on the characterisations made by Sensor Products. It assesses the sensor product characterisations in a tactical context and supplies objects to Tactical Objects supporting the separation of sensor processing to generate object evidence from the processing of evidence for the purpose of tactical interpretation.

These components are intended to be developed in a way which allows their capability to be expanded through new and updated algorithmic processes. The suggested technique to achieve this is extensions (see the Component Extensions PYRAMID concept). Whilst not mandated, application of this technique is shown in the examples throughout this PYRAMID concept.

Sensor Products may be extended to use different algorithms supporting different sensor data types and different processing techniques, for example an image processing algorithm that identifies particular features within an image. Data Fusion may be extended to use different fusion algorithms supporting different evidence types and fusion techniques, while Tactical Objects may be extended to use different algorithms used in the analysis of object relationships and behaviours. These components may also be data-driven providing configuration flexibility and ease of upgrade, for example in the range of template patterns used in the analysis of data, evidence and objects at their respective levels of abstraction.

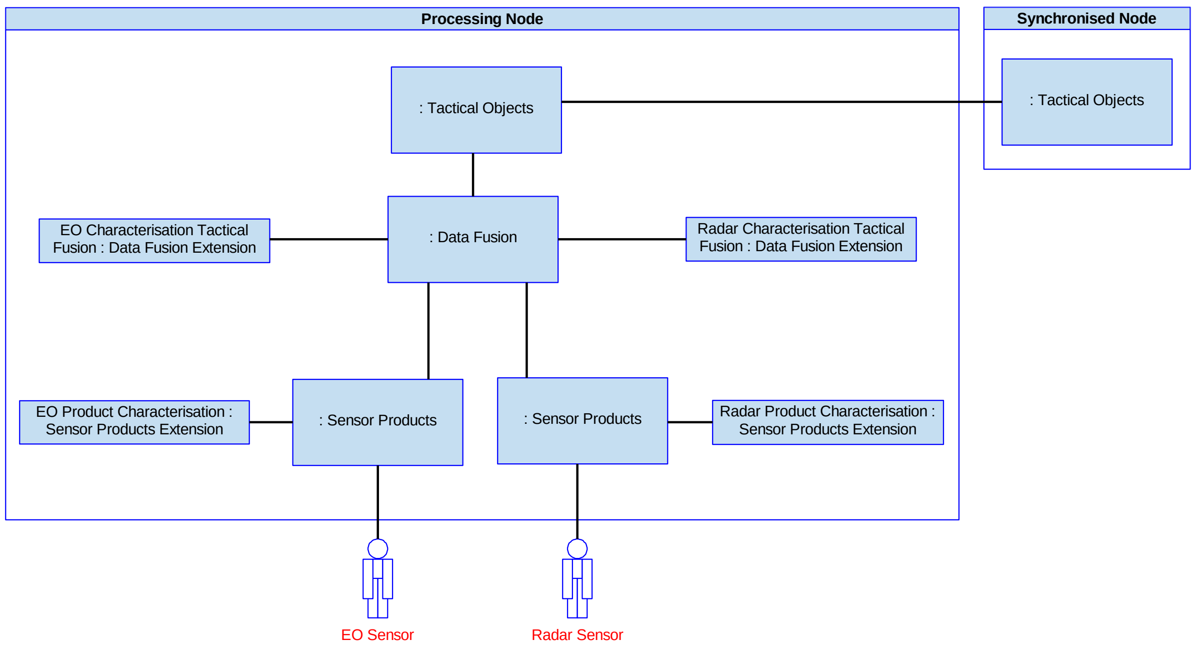

Processing Node
Synchronised Node
: Tactical Objects
: Tactical Objects
: Data Fusion
EO Characterisation Tactical Fusion : Data Fusion Extension
Radar Characterisation Tactical Fusion : Data Fusion Extension
EO Product Characterisation : Sensor Products Extension
: Sensor Products
: Sensor Products
Radar Product Characterisation : Sensor Products Extension
EO Sensor
Radar Sensor

Figure 96: Radar and EO System Example

Figure 96: Radar and EO System Example illustrates a PYRAMID deployment with two nodes: a processing node, which is responsible for sensor data handling to maintain a view of objects in the battlespace; and a synchronised node, which is synchronised to the processing node view of the battlespace, but does not do any sensor data handling.

The synchronised node only contains a Tactical Objects component. This simplifies this deployment as it only contains a view of objects in the battlespace, rather than a view of the objects and the lineage of their associated sensor characterisations.

The processing node also contains a Tactical Objects component, but this node is also responsible for the processing of sensor data including the tactical interpretation of that data, so it has been deployed with two instances of Sensor Products and an instance of Data Fusion.

The EO processing and radar processing elements of this could be reused on any Exploiting Programme that utilises those sensor data types. This should reduce the cost of PYRAMID deployments, as it allows reuse of an integrated set of components.

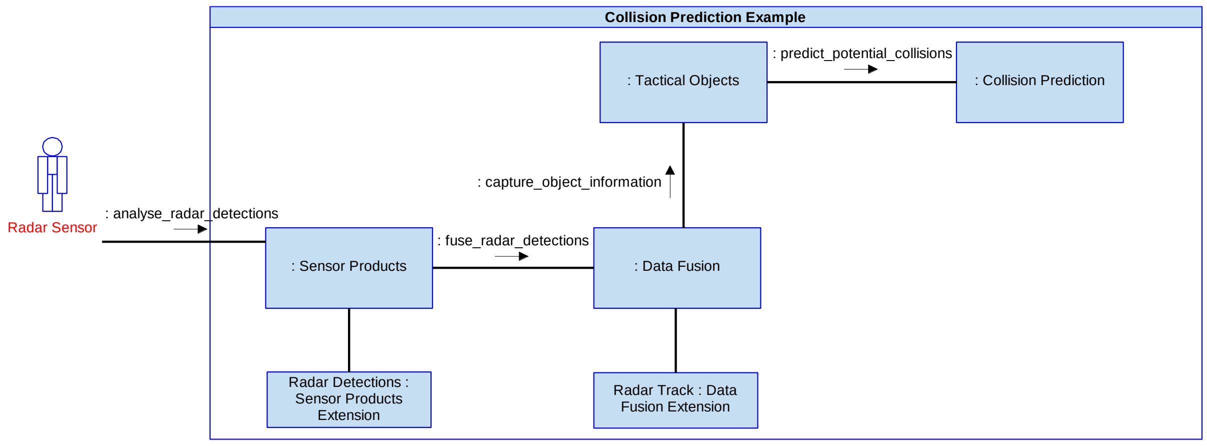

Collision Prediction Example
: Tactical Objects
: predict_potential_collisions
: Collision Prediction
: capture_object_information
Radar Sensor
: analyse_radar_detections
: Sensor Products
: fuse_radar_detections
: Data Fusion
Radar Detections : Sensor Products Extension
Radar Track : Data Fusion Extension

Figure 97: Collision Prediction Example

Figure 97: Collision Prediction Example illustrates a particular use of Sensor Products to support collision prediction monitoring. In this particular example Sensor Products is extended with a Radar Detections extension enabling the analysis and characterisation of radar sensor data. Data Fusion is extended with a Radar Track extension which receives the radar characterisation data from Sensor Products and provides a tactical interpretation of the evidence generating a single object track. This object information is captured by Tactical Objects, which is the component that maintains knowledge of objects in the battlespace and their relationships. Tactical Objects provides Collision Prediction with the relevant object positions and speed information since this meets the specified criteria important to Collision Prediction and where action may be required to avoid collision.

Designing the Sensor Products and Data Fusion components to be extended allows them to be tailored to a particular deployment, whilst separating them from knowledge of object relationships and behaviour allows nodes within a deployment to just consider the overall tactical picture.

A.2.16.6 Separation of Data Handling and Control

For Exploiting Platforms where variable control of sensor data handling and the extraction of meaning from that data is required, Tactical Objects, Sensor Products and Data Fusion are coordinated by the Sensor Data Interpretation component. For more detail see the Sensor Data Interpretation IV. In this IV a requirement to deliver interpretations is placed upon Sensor Data Interpretation which could have many possible solutions; Sensor Data Interpretation selects the most appropriate solution and enact the relevant control to deliver the solution.

Other platforms may not have the need to include variable control in this way, such as where object position prediction is required to be carried out every time a radar detection is received. In such cases a fixed or data-driven solution, which is continually being applied, may be more appropriate.

Sensor Data Interpretation may also coordinate Sensor Products and Data Fusion to extract meaning from stored sensor data, when the interpretation does not need to be done in real time at the point that the data is acquired.

Separating Sensor Data Interpretation from Sensor Products, Data Fusion and Tactical Objects creates flexibility when architecting deployments which require variable control of processing, while still enabling Exploiting Platforms which are of higher criticality to be more easily certified. It also allows the architecture to be extended to increase capability in terms of new processing algorithms, while still allowing flexible control of the processing algorithms to be applied during operation.

A.2.16.7 Tactical Information Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Tactical Information Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept does not identify any generic component behaviours in relation to tactical information.	The PRA does not define any generic component responsibilities or services in relation to tactical information.
Specific PRA Component support for the PYRAMID Concept	The PRA separates out areas of functionality associated with the handling of sensor data, from the tactical information derived from that data. The PRA separates out the following areas of functionality with regards to sensor data: • The provision of an interface to obtain measurements from sensors. • The data measured by a sensor, data derived or extracted from such a measurement, and the characterisation of such data. The PRA separates out the following areas of functionality with regards to the coordination of sensor, and sensor data processing, resources: • The coordination of sensor resources to measure properties of the environment. • The coordination of resources for the purpose of extracting meaning from data produced by sensors. The PRA separates out the following areas of functionality that relate to tactical information: • The evaluation and combination of evidence of potential objects to provide an understanding of real-world entities. • The knowledge of objects that exist, or are assumed to exist, within the battlespace, their relationship to each other and their behaviour. • The determination of the level of risk posed by a tactical object to another tactical object. • The calculation or estimation of the spectral signature of objects. • The evaluation of a subject's observability by an observer. • The assessment of an object's vulnerability to an aggressor's capability. • The knowledge of vehicle performance parameters for various vehicle configurations and modes of operation. • The prediction of objects spatial trajectories. • The knowledge of geographical features, including their location, characteristics and the relationships between them.	The PRA defines the following components whose role relate to a particular aspect of sensor data handling: • Sensors • Sensor Products The PRA defines the following components whose role relate to a particular aspect of sensor data, and sensor data processing, resources: • Sensing • Sensor Data Interpretation The PRA defines the following components whose role relate to a particular aspect of tactical information: • Data Fusion • Tactical Objects • Threats • Signature • Observability • Susceptibility • Vehicle Performance • Trajectory Prediction • Geography
Exploiter considerations	The following tactical information related considerations are a matter for individual Exploiting Programmes: • The choice of which PRA components to include according to the particular aspects of sensor data capture, sensor data processing and tactical information that are important to the Exploiting Programme, e.g. support for static or dynamic data interpretation requirements. • The approach to achieving the required specialisations across all sensor data handling and tactical data components, for example, to handle different types of sensors and sensor data, and different tactical entities. • The specific approach to system partitioning, for example the use of component instances and variants to manage tactical information across multiple platforms.	N/A

Table 17: Tactical Information PYRAMID Concept Summary

A.2.17 Test

A.2.17.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

- Health Management

Read in conjunction with

- Capability Management
- Interaction with Equipment
- Resource Management

A.2.17.2 Introduction and Scope

This PYRAMID concept explains how the ability to support testing is provided by the PRA. It describes what is meant by testing, and how the components, including the Test component, support different types of test at various levels of system capability.

A.2.17.2.1 What is a Test, or Testing?

In this PYRAMID concept, test means a procedure or method intended to establish the state, performance and capability of something.

Testing supports the identification of the current capability of a system in line with the Capability Management PYRAMID concept by determining the capability of the hardware or equipment that support the system capability. Although testing can be triggered at any level, the actual test is carried out on a resource. Testing can be used to force the determination of possible capability limits prior to use, or to achieve awareness of a loss of capability or anomaly. Testing may also be instigated in response to the need to effectively determine likely causes of a recognised loss of capability or an anomaly; this is described more broadly in the Health Management PYRAMID concept.

A.2.17.2.2 Why is Testing Important?

Testing is important for determining the capability of a system, and where testing reduces the ability of the system to fulfil other capabilities, it can be coordinated and managed in order to minimise system non-operational time.

A.2.17.2.3 Scope of this PYRAMID Concept

This PYRAMID concept covers testing for capability and function; it excludes:

- Testing during software development (verification of requirements).
- Testing during system integration.
- Test harnesses and facilities (e.g. rigs).
- Flight test (including associated ground tests).
- Validation of the successful loading of software.

- Breakdown/build testing (including upgrades).

A.2.17.3 Overview

Testing can be undertaken at any level of a system's hierarchy or abstraction. No single component is responsible for all aspects of testing. Each component is responsible for testing its own area of capability, following a standardised approach; this allows the component's status to be determined independently.

However, the Test component manages and coordinates testing:

- Across multiple components (e.g. at system level).
- In interfacing equipment or infrastructure (including some built in tests) where those tests reduce system capability, consume additional resources (e.g. electrical power) or may pose safety/security risks.

A.2.17.4 Testing of Component Capability

Each component is responsible for assessing its own capability. However, components depend on each other for their capability, or, in the case of Resource layer components, on the resources they control. Components can request better (more specific, or more certain) information about the capabilities they depend on and, for Resource layer components, that may be achieved by triggering a test of the resource. Equipment can also self-trigger testing (e.g. as part of the start-up procedure). The need for information can come from any level of the system, but the test to provide the information is carried out on the equipment.

A component that represents an equipment's function understands how to obtain information about that function, which could include running a Built in Test (BIT). Therefore, BIT, as a test mechanism integral to equipment, will mainly be initiated by Resource Layer components (see the Interaction with Equipment PYRAMID concept) or managed internally to the equipment (for example, running a Power-Up BIT (PBIT) as part of the start-up procedure). This does not exclude the use of BIT as part of a prompted or coordinated test resulting from an Action layer component requesting information; in such a case the initiating component does not know how the required test is being achieved, i.e. it does not know if it is done using equipment BIT or another method.

If a test will interfere with another use of a resource, the conflict will be identified to Conflict Resolution for resolution in line with the Dependency Management PYRAMID concept. A test request must be rejected if it would cause the exceedance of a limit.

A.2.17.5 Management of Testing

The Test component knows which tests will provide which information and will manage testing across multiple components (e.g. at system level). This allows a component to support systematic tests without requiring additional effort, test specific software or an understanding of mode of use. Test will initiate (trigger) the different tests or the different aspects of a test, in line with the requirements of the test procedure.

A.2.17.6 Example: Performing an IBIT

Figure 98: Performing IBIT shows how components involved in a test might interact.

Health Assessment has identified the need for additional information to diagnose more precisely the effects on capability and physical assets.

The Test component identifies the tests that will provide this additional information and executes them. The tests need to be coordinated on several resources; such synchronisation may be required to minimise system non-operational time, or to sequence tests to ensure that the effect of one test does not unintentionally impact the results of another.

The tests run by the resource components may be invoked internally or by invoking BIT on an item of connected equipment. Where a demand on a resource component conflicts with other activities, that component will identify the conflict to Conflict Resolution for brokering and arbitration, in line with the Dependency Management PYRAMID concept. This is shown in this example for access to the fuel distribution system.

The results of the testing are reported and acted upon as required. Results may also take the form of health data which is used by Anomaly Detection and Health Assessment, as described in the Health Management PYRAMID concept.

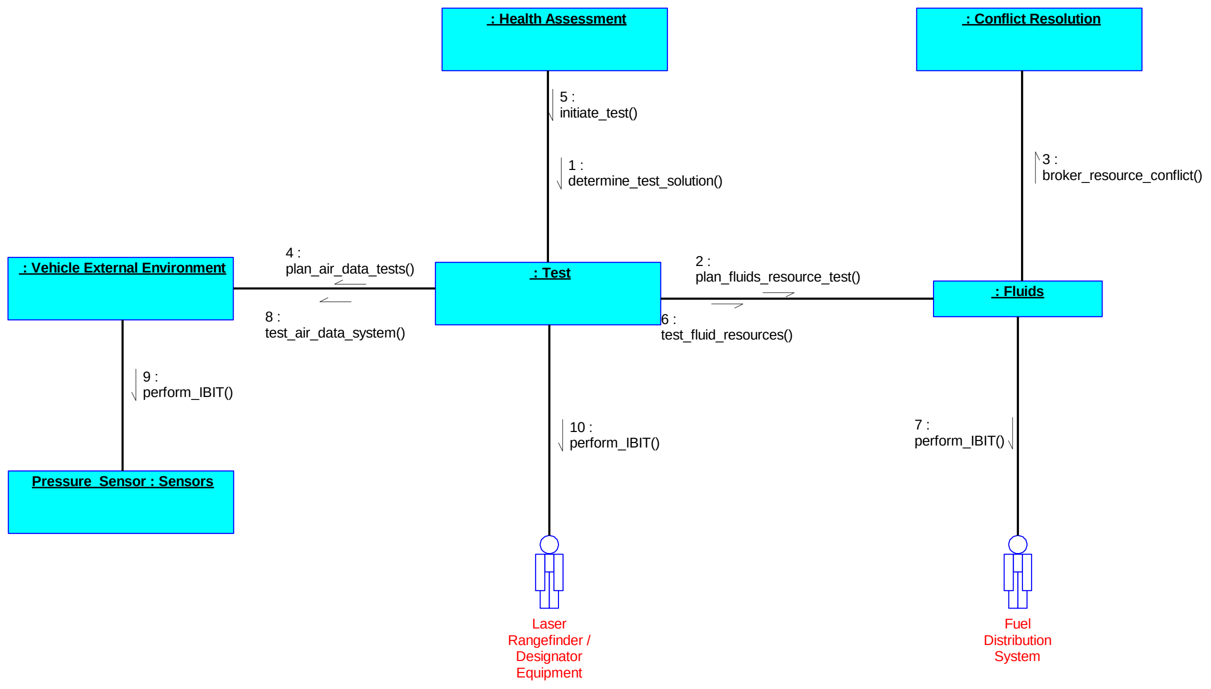

: Health Assessment
: Conflict Resolution
5 : initiate_test()
1 : determine_test_solution()
3 : broker_resource_conflict()
: Vehicle External Environment
4 : plan_air_data_tests()
: Test
2 : plan_fluids_resource_test()
: Fluids
8 : test_air_data_system()
6 : test_fluid_resources()
9 : perform_IBIT()
10 : perform_IBIT()
7 : perform_IBIT()
Pressure_Sensor : Sensors
Laser Rangefinder / Designator Equipment
Fuel Distribution System

Figure 98: Performing IBIT

A.2.17.7 Example: Equipment Health Test Result Interpretation

Figure 99: Equipment Health Test Result Interpretation shows how multiple components can interpret or process the results of an equipment test. This example shows one way that the PYRAMID concept could be applied. It involves external equipment whose various aspects are managed through multiple resource components, in line with the principles described in the Interaction with Equipment PYRAMID concept.

A test requestor (not shown, but for example the Test component or an operator) initiates requirements to perform IBIT on the Sensor Equipment. Sensors allocates the resource for the test and initiates the IBIT. Sensor Equipment performs an IBIT and generates an error code which is picked up by Anomaly Detection. Anomaly Detection confirms that the error code is related to an anomaly and informs Sensors, Mechanical Positioning and Health Assessment.

Mechanical Positioning reads the error code and assumes its capability is unchanged because the error code is not related to its capability.

Sensors reads the error code and recognises it as related to its capability. Health Assessment analyses the error code along with other health data and determines the cause of the failure (e.g. the lens of an IR sensor is blocked and is not able to provide the required focus). This gives Sensors more details on its affected capability and allows it to take more specific actions.

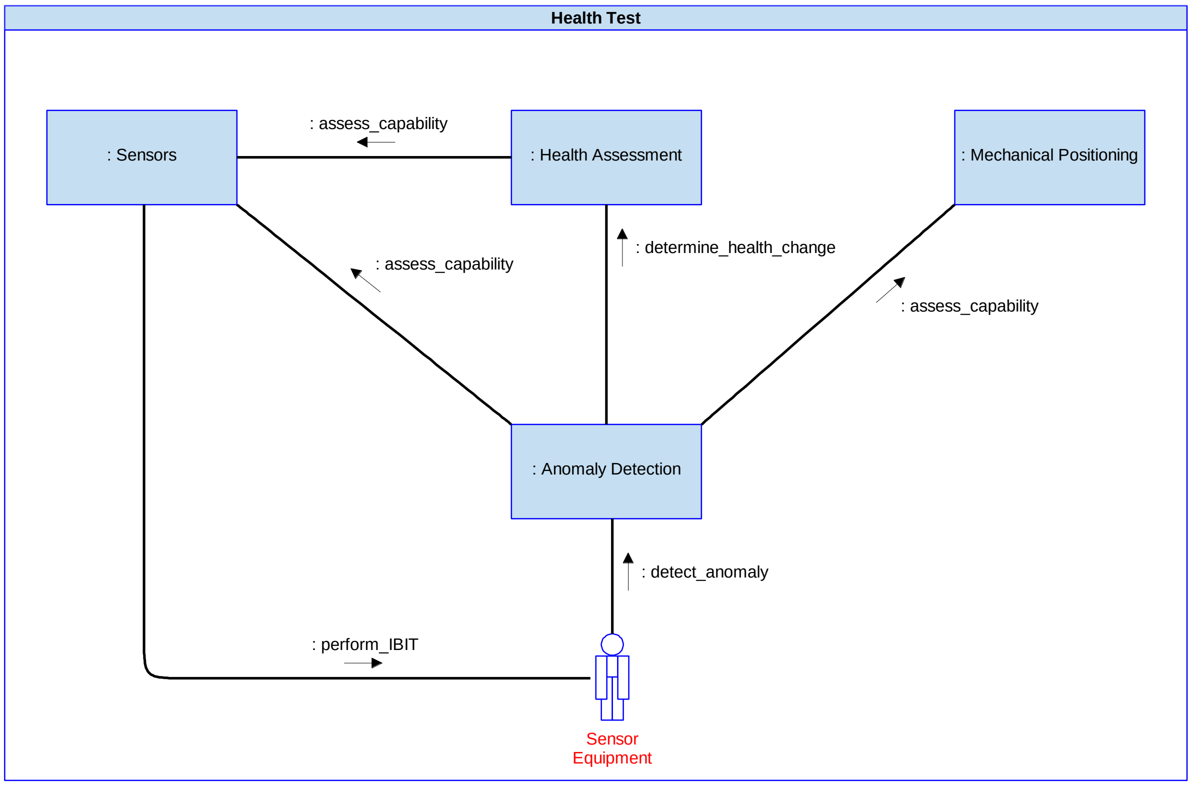


Figure 99: Equipment Health Test Result Interpretation

A.2.17.8 Test Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Test Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	Any component may identify a need for information that results in a requirement to perform a test, e.g. the need for additional capability evidence. The PYRAMID concept identifies the following generic component behaviours in relation to test: • Each component is responsible for assessing and testing its own area of capability. Where the need to execute a test conflicts with another use of a resource, a process of brokering and arbitration is used to resolve the conflict.	The responsibilities of individual components in respect of assessing and reporting capability is described in the Capability Management PYRAMID concept. Where a component provides an interface to an external equipment, a corresponding service may or may not be defined by the PRA: Where the interface is not at the PRA scope boundary, and the function provided by the external equipment could have been provided by a PYRAMID component, then an associated service will be included in the PRA. Where the interface corresponds to the PRA scope boundary, services will not be defined by the PRA (see PYRAMID Technical Standard, Ref. [1], section Services Crossing the PRA Scope Boundary). For more information see the Interaction with Equipment PYRAMID concept.
Specific PRA Component support for the PYRAMID Concept	The PRA separates the following areas of functionality that relate to test: • An understanding of the tests that can be run to establish the condition of part of a system and the coordination of test resources to perform tests.	The PRA defines the following component with a specific role in relation to Test: • Test
Exploiter considerations	This PYRAMID concept does not identify any specific Exploiting Programme considerations.	N/A

Table 18: Test PYRAMID Concept Summary

A.2.18 Use of Communications

A.2.18.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Data Exchange

A.2.18.2 Introduction and Scope

This PYRAMID concept explains how the PRA has been designed to support the use of communications.

A.2.18.2.1 What is Use of Communications?

This PYRAMID concept is intended to illustrate how components with differing levels of communications awareness interact with components and systems, on the same or different platforms, both directly and by using communication infrastructure. It is not intended to mandate the underlying communications architecture, specific protocol, or formatting approach which may be adopted; this will be determined by the Exploiting Programme.

Components are by default agnostic of other components and systems' use, intent and location. They can send and receive signals but in most cases do not know where the signals come from or go to, or the route via which those signals travel. Where necessary, the routing of these signals will make use of communications capability, however by default the originating and destination components will not be aware they are doing so. In some cases, however, it may be appropriate for a component to have some level of awareness of the need for communications.

The subsequent sections of the PYRAMID concept list the different levels of communications awareness that may exist in components, and outlines the impact on those components' responsibilities.

A.2.18.2.2 Why is the Use of Communications Important?

This PYRAMID concept is important because it defines how communications between a component and other components or systems, which are on the same or different platforms, are managed.

A.2.18.2.3 Exclusions to this PYRAMID Concept

This PYRAMID concept does not cover the process of managing multiple nodes, responding to loss of communications or security of communications. Refer to the following PYRAMID concepts for further information:

- Multi-Vehicle Coordination PYRAMID concept
- Health Management PYRAMID concept
- Capability Management PYRAMID concept

- Cyber Defence PYRAMID concept.

A.2.18.3 Overview

This PYRAMID concept explains how components with differing levels of communication awareness can communicate and how this communication is managed. Components can send and receive signals but the level of information they have on where the signal goes or the route the signal takes determines the components communication awareness. Components can be:

- **Communication agnostic components**: a component that does not directly generate dependencies that are placed on communications capability components.
- **Communication aware components**: a component whose level of reasoning about the concept of multiple nodes, or component interactions between nodes, is such that it directly generates dependencies that are placed on communications capability components.
- **Communications capability components**: a component that contributes to the provision of communications between nodes, either through the provision of communications infrastructure, or the delivery of data over that infrastructure.

A.2.18.4 Levels of 'Communications Awareness' within Components

The term communication is used within this PYRAMID concept to mean the process by which a signal is transmitted from a node, within a system, to another node or system. The level of communication awareness can be determined based on the understanding that a component has about the need for that signal.

A.2.18.4.1 Communications Agnostic

Components are assumed by default to be communications agnostic and not directly generate dependencies that are placed on communications capability components. Requirements placed by a communications agnostic component may be subsequently determined to have communications-related dependencies, but these are ‘invisible’ to the communications agnostic component. This means that where a component sends a message to another component, it is not aware of the other's location (in terms of communications nodes), or the requirements for the connection between them. Equally, when a communications agnostic component receives a message, it is not aware that it is being accessed by a component with which it is not physically co-located. However, the design of a communications agnostic component needs to accommodate for the possible latency of signals as a result of components not being co-located. This could be done through data driving or by allowing different qualities of service, based on the latency associated with a specific deployment or system configuration.

A.2.18.4.2 Communications Aware

Whilst components are communications agnostic by default, in some cases a component may have a need to be communications aware.

Where a component needs to send a message, for example, the component may be able to identify a node associated with the remote recipient to the communications capability components, without understanding any details of how that connection is achieved that fall outside its subject matter.

These components will consume services provided by the communications capability components such as:

- Placing constraints on communications activity (such as prohibiting traffic to a particular node due to security concerns).
- Requesting a connection to be established (or made available) to a particular recipient.
- Requesting information on the state (including availability) of connections.

The Tasks and Objectives components and some action components are likely to be aware that an activity requires connecting physically remote components. In some cases, communications aware components may need to understand (and even mandate in detail) the requirements of the connection.

A.2.18.5 Communications Capability Components

Communications capability components are those components which form part of the communications capability itself, and whose primary collective responsibility is to provide communications for other components. This consists of:

- Communications infrastructure management (which establishes and maintains communication channels), which is provided by:
    - Networks (responsible for the set-up, management and optimisation of communication network topologies),
    - Communication Links (responsible for the establishment, maintenance and optimisation of communication links between nodes),
    - Network Routes (responsible for the per-hop delivery of data between nodes within a network),
    - Communicator (responsible for the management of resources that enable the transmission and reception of communication signals).
- Data delivery (which transfers information along the channels provided by the infrastructure), which is provided by:
    - Human Interaction (responsible for the management of resources that specifically enable communication between humans),
    - Information Brokerage (responsible for the understanding of the legitimate information needs of participants and the instigation and oversight of the required, and allowed, information exchanges),
    - Semantic Translation (responsible for the translation between the semantics of participants involved in an information exchange).
    - Data Distribution (responsible for the distribution and reception of data including the collation and formatting of data required to support the delivery),

Note that the communications capability components will also be users of communications, in order to communicate and agree communications capability.

An indicative outline of the components forming the communications capability, and the role each plays, is illustrated in Figure 100: Communications Capability. Note that it is not exhaustive to all communications activities (for example, as illustrated in the Human Communications IV, a direct human interaction through communications would draw in the HMI Dialogue and Information Presentation components).

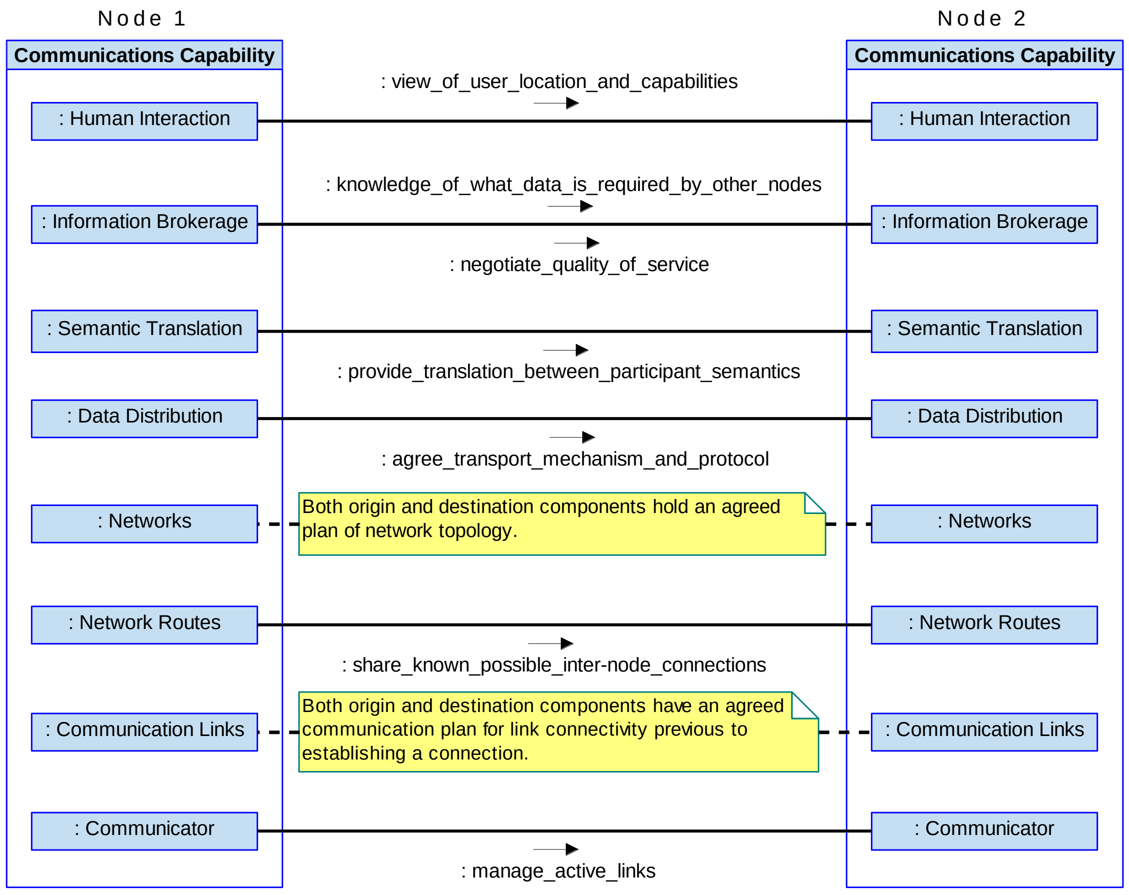

Node 1
Node 2
Communications Capability
Communications Capability
: Human Interaction
: Information Brokerage
: Semantic Translation
: Data Distribution
: Networks
: Network Routes
: Communication Links
: Communicator
: view_of_user_location_and_capabilities
: knowledge_of_what_data_is_required_by_other_nodes
: negotiate_quality_of_service
: provide_translation_between_participant_semantics
: agree_transport_mechanism_and_protocol
Both origin and destination components hold an agreed plan of network topology.
: share_known_possible_inter-node_connections
Both origin and destination components have an agreed communication plan for link connectivity previous to establishing a connection.
: manage_active_links

Figure 100: Communications Capability

Note that in practice, a communications flow would pass down the 'ladder' of components in the originating node before passing back up in the destination node, in accordance with a common plan agreed between the originator and recipient.

The communications capability components are responsible for providing communications, by carrying out activities covering the planning, operation and maintenance of communications networks.

A.2.18.5.1 Communications Planning

Where an extant communications link is available, the communications capability will be provided by the Data Distribution component. Where it is identified that the Data Distribution component cannot currently meet a communications requirement, a new communications link will be established in accordance with the Network Initialisation IV.

A.2.18.5.2 Communications Operation

For examples of communications operations for transfer of data, cryptography, management of established communications channels, or real-time communication between humans, refer to the following interaction views:

- Data Transfer IV
- Cryptographic Management IV
- Connection Management IV
- Network Initialisation IV
- Human Communications IV

A.2.18.5.3 Communications Maintenance

The ongoing performance monitoring, evaluation and reconfiguration of communications resources is managed by the appropriate communications capability components, in accordance with their defined responsibilities and as described in the Capability Management and Resource Management PYRAMID concepts (see the Data Transfer IV for an example).

A.2.18.6 Use of Communications Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Use of Communications Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	Components in general have no knowledge that their required interactions within the system may be dependent upon underlying communications, and their subject matters are entirely agnostic of the existence or nature of any such communications. However, where it is in keeping with a component's defined subject matter to understand, in some respect, the multi-node nature of a system, then components may have communication related dependencies. The PYRAMID concept does not identify any generic component behaviours in relation to use of communications.	The PRA does not define any generic component responsibilities or services in relation to the use of communications.
Specific PRA Component support for the PYRAMID Concept	The PRA separates the following areas of functionality in respect of communications capability: • The management of resources that specifically enable communication between humans. • The understanding of the legitimate information needs of participants and the instigation and oversight of the required, and allowed, information exchanges. • The translation between the semantics of participants involved in an information exchange. • The distribution and reception of data including the collation and formatting of data required to support the delivery. • The set-up, management and optimisation of communication network topologies. • The per-hop delivery of data between nodes within a network. • The establishment, maintenance and optimisation of communication links between nodes. • The management of resources than enable the transmission and reception of communication signals.	The PRA defines the following components whose roles relate to the use of communications: • Human Interaction • Information Brokerage • Semantic Translation • Data Distribution • Networks • Network Routes • Communication Links • Communicator For more information on the Information Brokerage, Semantic Translation and Data Distribution components also see the Data Exchange PYRAMID concept.
Exploiter considerations	The following communication capability related considerations are a matter for individual Exploiting Programmes: • The definition of a communications architecture and choice of communication protocols. • The approach to ensuring communications security.	The PRA does not mandate a specific approach in respect of these considerations.

Table 19: Use of Communications PYRAMID Concept Summary

A.2.19 Data Exchange

A.2.19.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

- Use of Communications
- Component Connections
- Component Extensions

A.2.19.2 Introduction and Scope

The Data Exchange PYRAMID concept explains the PRA support for the capability associated with the exchange of data and information. The need to exchange data between elements of a system, or between systems, introduces a number of exploitation considerations. Note that for the purpose of this PYRAMID concept, these systems or system elements will be referred to as participants, and cover the following capability:

- The determination of what information is required by participants and whether they are allowed to have it.
- How the data is formatted and suitably protected to ensure effective and secure data exchange.
- Whether the data has to be interpreted or translated to be meaningful to the recipient, bridging any semantic gap between the participants.

This PYRAMID concept explains how the PRA has been developed to support these aspects of Data Exchange, noting that the broader scope of the use of communications to support the exchange of data is covered in the Use of Communications PYRAMID concept.

For a simple system the considerations above may be achieved by the use of bridges; in more complex cases such considerations may go beyond the expected scope of a bridge, where the aspects of what may be included in a bridge is defined in the Component Connections PYRAMID concept. For these more complex cases the PRA provides a number of specific components that enable the active and dynamic management of the exchange, for example:

- Where the rules and restrictions for exchanging data between participants varies over time (e.g. the priorities and permissions) or are complex (e.g. in respect of collation and coordination).
- Where the semantic gap cannot be statically defined and needs to be dynamically determined for data exchange to begin.
- Where the data exchange requirement is sufficiently complex that the semantic translation cannot be carried out without putting too much capability into the bridge (e.g. where the decision to carry out data exchange requires an understanding of the data, such as the decision to exchange a sensor track depending on the quality of that track).

This PYRAMID concept does not define the specific methods of data delivery (e.g. specific exchange protocols, data formats and encapsulation methods). This PYRAMID concept focuses on the separation of the concerns and responsibilities in relation to the considerations described above. Specifically the PYRAMID concept discusses the role of the Information Brokerage, Semantic Translation and Data Distribution components in enabling the process of data exchange. Throughout this PYRAMID concept, the potential application of extensions for the implementation of these components is discussed (see the Component Extensions PYRAMID concept for an explanation of component extensions).

A.2.19.2.1 Definition of Data Exchange Terms

Within this PYRAMID concept, the terms data, information and participant are distinguished as defined in the following sub-sections.

A.2.19.2.1.1 Definition of Data

Within this PYRAMID concept the term "data" refers to a collection of values (that are being stored or transferred) without connotation to an understanding of the meaning of those values. For example, the term data refers to items that are being distributed, but the distribution mechanism is unable to interpret the data or understand its use.

A.2.19.2.1.2 Definition of Information

"Information" is knowledge (generated, received, manipulated or stored in the system) that has been derived from, or used to produce, data.

A.2.19.2.1.3 Definition of Participant

A component, node, partition or other system/system element that provides or receives information is a participant in the exchange. It is not intended that a participant will be a human operator, see the Human-Machine Interface PYRAMID concept for information on when a human is involved in the communication.

A.2.19.2.2 What is Exchange?

Exchange is the method by which data or information is passed between a PYRAMID deployment and another participant.

It consists of two elements: distribution and receipt. The process of distributing data may include the approval for transfer prior to transmission. For the process of receipt, when data is received by the participant there may be additional checks for approval prior to delivery to components for exploitation. Data and information exchanges can only take place between a participant capable of distribution and a participant capable of receipt.

A.2.19.2.3 Why Is Exchange Important?

The exchange of data and information is important as it could affect the fulfilment of the mission objectives and enables interoperability. There are multiple ways exchanges between participants can be achieved and it is important that the data being distributed (or received) is allowable and will not interfere with activity fulfilment.

At the point of exchange, it may be possible to mitigate for data which is untimely from a low trust source.

A.2.19.3 Overview

The PYRAMID concept outlines how the Information Brokerage, Semantic Translation and Data Distribution components interact in order to provide the distribution and receipt of data. These component interactions are illustrated with examples of Voice over Internet Protocol (VoIP), Data Distribution Service (DDS) and an exchange with a STANAG 4559 information system.

A.2.19.4 Distribution

This section focuses on how data and information is sent from a participant. For how information is received and exploited by a participant see the Receipt section of this PYRAMID concept.

Distribution could be done via different communications protocols which will follow an adaptation of a standard pattern, see the Distribution Variations section of this PYRAMID concept.

The standard pattern includes three components, Information Brokerage, Semantic Translation and Data Distribution, as illustrated in Figure 101: Standard Pattern.

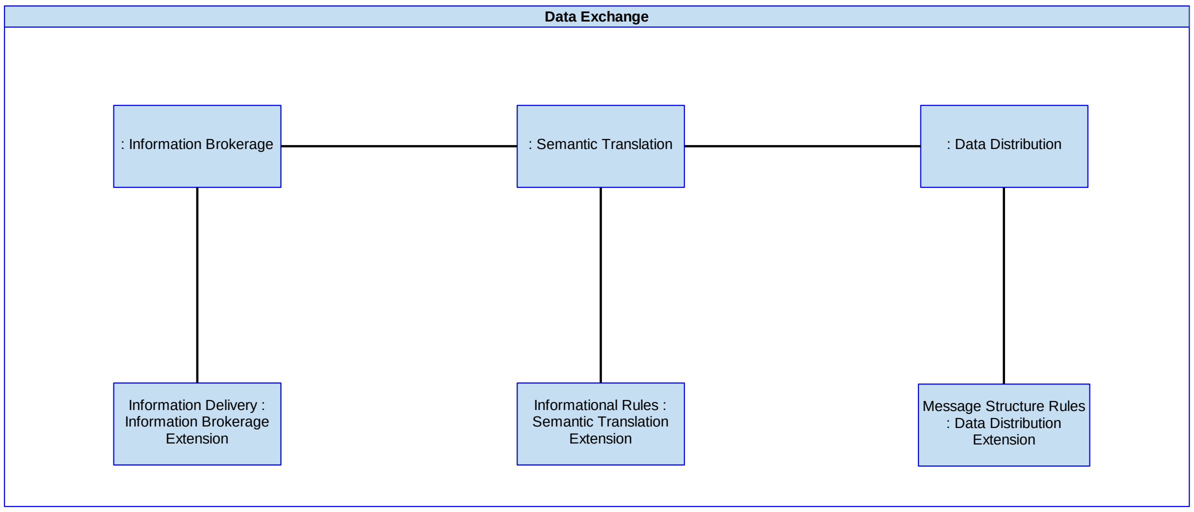

Data Exchange
: Information Brokerage
: Semantic Translation
: Data Distribution
Information Delivery : Information Brokerage Extension
Informational Rules : Semantic Translation Extension
Message Structure Rules : Data Distribution Extension

Figure 101: Standard Pattern

Information Brokerage understands what information, produced by the system, has been requested by other participants and tries to fulfil these requests. It will perform checks to ensure the information is allowed to be distributed to the requesting participants and identify the need for the information being exchanged to be represented in a particular way (e.g. represented as JPEG or PNG, represented using a particular reference frame and axis system, or represented at a particular security classification). Any conversion of the information to the required representation is not carried out by Information Brokerage; it will be carried out by the component that understands how to perform the conversion, which is typically the originating component. In contrast, the formatting of the data to facilitate its transfer and delivery is performed by Data Distribution (e.g. changing how the data is encoded in order to reduce its susceptibility to corruption). Once Information Brokerage has identified that a piece of information is allowed to be distributed it will indicate this.

Semantic Translation understands the needs of participants involved in the exchange of a specific piece of information, i.e. informational rules required to be adhered to in the distribution of data.

The informational rules may be captured in extensions. Examples of these can be seen in the Distribution Variations section of this PYRAMID concept. Once Semantic Translation has applied the correct informational rules for the distribution it will pass the appropriate data for delivery.

Data Distribution knows that a piece of data requires distribution to a recipient via a specific distribution method. Data Distribution understands the methods of distribution and how to prepare data for the distribution. This includes ensuring that the data is suitably protected and error free before distribution, for example by carrying out error checks or making use of the capabilities of components such as Cryptographic Materials and Cryptographic Methods for protection (such as encryption). The functionality of preparing the data may be performed by extensions which understand the message structure rules for that distribution method; examples can be seen in the Distribution Variations section of this PYRAMID concept. Once the data has been prepared for distribution Data Distribution will initiate the distribution.

For further representations of this template, see the Tactical Exchange IV, the Request for Information IV or the Distribution Variations section of this PYRAMID concept.

The following section shows how this pattern may be implemented using specific protocols.

A.2.19.5 Distribution Variations

A.2.19.5.1 Voice over IP

Figure 102: VoIP Voice illustrates how the standard pattern would be used to represent a VoIP voice distribution system.

Voice over Internet Protocol (VoIP) is a specific method for the delivery of voice communications. In this example, the types of extension used are VoIP delivery for the Information Brokerage component, VoIP Informational Rules (codec) for the Semantic Translation component and RDP/SIP Structure Rules for the Data Distribution component.

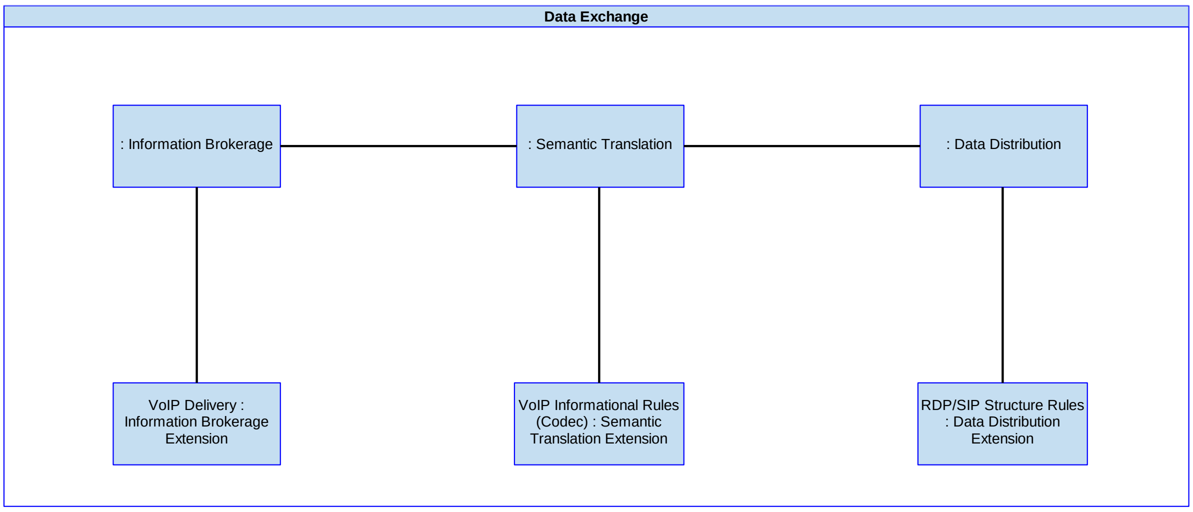

Data Exchange
: Information Brokerage
: Semantic Translation
: Data Distribution
VoIP Delivery : Information Brokerage Extension
VoIP Informational Rules (Codec) : Semantic Translation Extension
RDP/SIP Structure Rules : Data Distribution Extension

Figure 102: VoIP Voice

A.2.19.5.2 Data Distribution Service

Figure 103: Data Distribution Service (DDS) illustrates how the standard pattern would be used to represent a Data Distribution Service (DDS).

DDS is a system-to-system standard, using a publish-subscribe pattern that allows real-time data exchange. In this example, the Semantic Translation component is not included as it does not perform any function. The types of extension used are Information Delivery for the Information Brokerage component and RTPS Structure Rules for the Data Distribution component. A representation of this can be seen in the Request for Information interaction view.

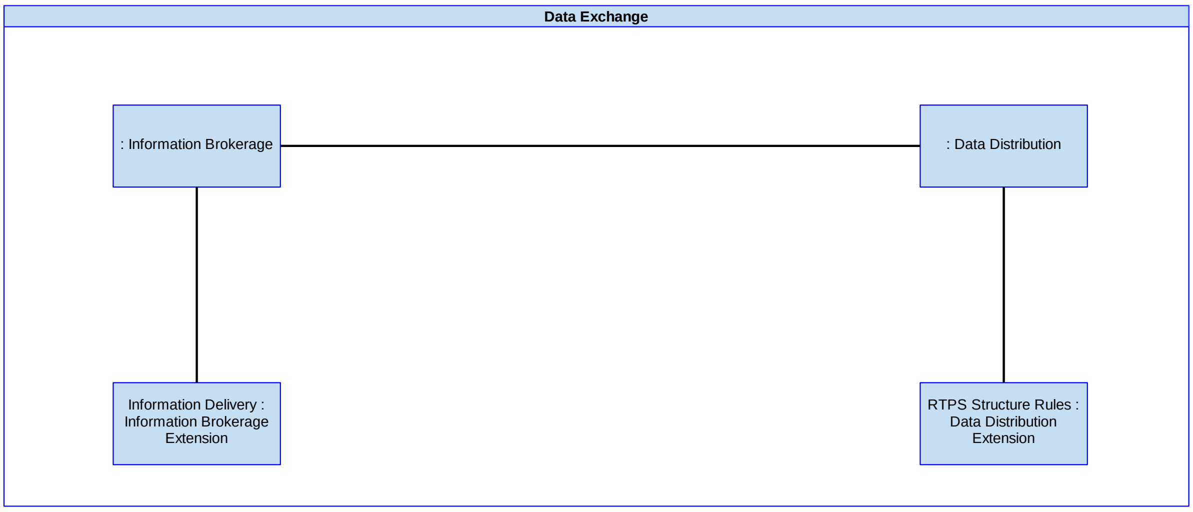

Data Exchange
: Information Brokerage
: Data Distribution
Information Delivery : Information Brokerage Extension
RTPS Structure Rules : Data Distribution Extension

Figure 103: Data Distribution Service (DDS)

A.2.19.5.3 STANAG 4559

Figure 104: STANAG 4559 illustrates how the standard pattern would be used to represent STANAG 4559.

STANAG 4559 is a standardised agreement to promote the interoperability of NATO member states' Intelligence, Surveillance and Reconnaissance (ISR) products managed by product libraries. In this example, the types of extension used are Information Availability for the Information Brokerage component, 4559 Interchange Rules for the Semantic Translation component and Network Structure for the Data Distribution component.

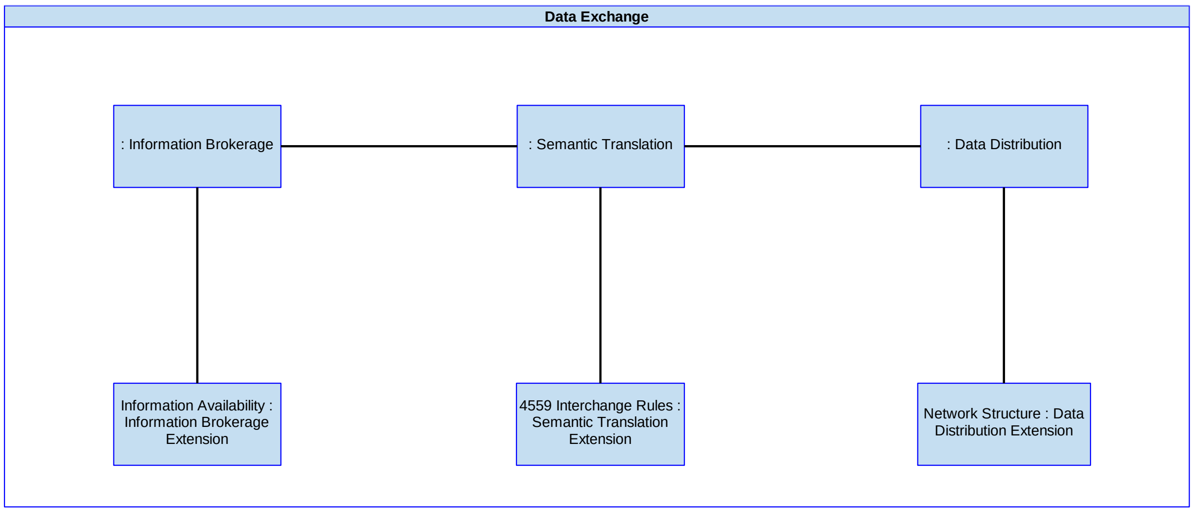

Data Exchange
: Information Brokerage
: Semantic Translation
: Data Distribution
Information Availability : Information Brokerage Extension
4559 Interchange Rules : Semantic Translation Extension
Network Structure : Data Distribution Extension

Figure 104: STANAG 4559

A.2.19.6 Receipt

This section focuses on how data and information is received from a participant, checked and then sent to the correct component for exploitation. For how data and information is sent see the Distribution section of the PYRAMID concept.

Similar to Distribution, and the examples shown in the Distribution Variations section, receipt will follow an adaptation of the standard pattern shown in Figure 101: Standard Pattern. Here the initial stage for receipt is at Data Distribution where the received data is checked for integrity, unpackaged and changed into the message structure for the participant by the Message Structure Rules extension. Once it has been formatted into a message structure to be understood by the participant, Semantic Translation is informed that a piece of data has been received. Semantic Translation will use its extension Informational Rules to understand what method has been used and how it can be used by the participant. If required, Semantic Translation will inform Information Brokerage that a piece of information is available and Information Brokerage will know which component is interested in that piece of information and therefore be able to ensure the information is exploited within the system.

An example of this, in which Information Brokerage is not required, can be found in the Tactical Exchange IV, where a Tactical Datalinks (TDL) Track is received.

A.2.19.7 Data Exchange Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Data Exchange Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	All PRA components are involved in the exchange of information according to the needs of their subject matter or demands on their subject matter. Such exchanges could be associated with any type of component interaction, such as the placement of requirements, as well as the necessary sharing of subject matter information. This may include exchanges with participants that are external to the PYRAMID deployment. The means by which data is exchanged between participants and the specific considerations for any exchange (semantics, structure, format or protection) depends on the nature of the participants and their relationship, and the nature of the information being exchanged.	All PRA components have defined provided and consumed services, consistent with their subject matter responsibilities and that enable interaction with other system elements. Data exchange between a PRA component and another system element is achieved using a bridge as described in the Component Connections PYRAMID concept. Bridges are used to perform the translations necessary to facilitate exchanges while enabling the component to remain independent of the structure and semantics of other system elements. In more complex cases, data exchanges are supported by a number of dedicated PRA components identified below.
Specific PRA Component support for the PYRAMID Concept	The PRA separates out the following areas of functionality that relate to data exchange: • The understanding of the legitimate information needs of participants and the instigation and oversight of the required, and allowed, information exchanges. • The translation between the semantics of participants involved in an information exchange. • The distribution and reception of data including the collation and formatting of data required to support the delivery.	The PRA defines the following components whose role relates to data exchange: • Information Brokerage • Semantic Translation • Data Distribution The PRA defines a number of components that deal with different aspects of communication. For more information see the Use of Communications PYRAMID concept.
Exploiter considerations	The following data exchange related considerations are a matter for individual Exploiting Programmes: • The specific mechanisms and protocols used in the delivery of data, including any associated security measures.	The PRA does not mandate a specific approach in respect of these considerations.

Table 20: Data Exchange PYRAMID Concept Summary

A.3 PRA Exploitation Principles PYRAMID Concepts

PRA Exploitation Principles are PYRAMID concepts that explain concepts that may be applied to a PYRAMID deployment in a manner consistent with achieving the PYRAMID benefits.

A.3.1 Component Connections

A.3.1.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

None

A.3.1.2 Introduction and Scope

This PYRAMID concept explains how component connections can be used between components to produce complex systems.

A.3.1.2.1 What is a Component Connection?

The PRA contains a set of components that form the building blocks of a deployment. Component connections define relationships between these components to allow them to work together as a system.

A.3.1.2.2 Why Are Component Connections Important?

Component connections are fundamental to the construction of a system of components. It is important to consider the component connections when describing system behaviour. The way components are connected makes an essential contribution to their ability to utilise one another's capability and information, whilst avoiding the pollution of subject matter within components.

A.3.1.3 Overview

The PYRAMID concept for component connections can be summarised as follows:

- Components are connected together using bridges, which bridge the *semantic gap* between the components.
- Bridges should be limited to a well-defined set of functions to map and translate between component services and avoid including inappropriate PRA component content.
- Counterpart relationships are a common component connection concept that captures the relationships between the information of the different components that describe different aspects of the same real-world object or concept.
- The behavioural dependencies between components can be expressed by specifying these relationships in a component interaction pattern.

- The use of standard service patterns for component connections will ease integration, while still providing a mechanism for predictable, complex and flexible behaviour.

A.3.1.4 Component Information Relationships

In order to be able to connect components together to create a working system, the information relationships between the components and how they can be implemented needs to be understood. There are two important concepts pertaining to these component relationships that can be used to design the component connections. These are:

- The use of bridges
- The concept of counterparting

A.3.1.4.1 Bridges

Components in the PRA are deliberately scoped to have no knowledge of other components within a system. This approach creates a set of well bounded and loosely coupled components, maximising the opportunity for reuse and minimising the impact of changes, easing maintenance overheads, as well as allowing components to be developed in isolation from one another.

Each component represents a distinct subject matter area that is defined in its own language. Consequently, there is no shared interface definition between PRA components. Instead, a deployment can use bridges to close the semantic gap, aligning the interfaces between components so that they are able to share information. A bridge is designed to achieve the component connection that is required by the system's needs.

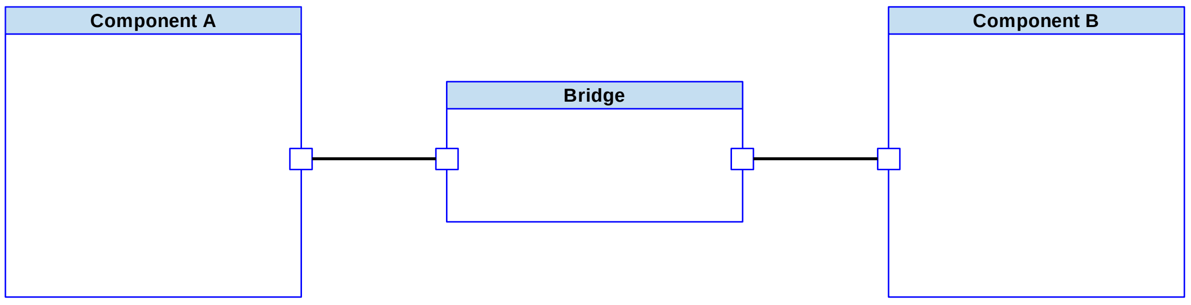


Figure 105: Basic Bridge Diagram

Figure 105: Basic Bridge Diagram shows a single bridge facilitating connection between two components (i.e. a one to one mapping of a consumed service to a provided service); however a bridge may also be used in a deployment to facilitate one to many connections (e.g. one component's consumed service mapping to two or more component's provided services) and even many to many connections.

A bridge defines which component services and information are connected to which other component services and information. To achieve this a bridge can perform the following three functions:

- Data type conversion (translating between the different formats and measurements used by different components).
- Semantic translation (translating the meaning of data in order to bridge the semantic gap between different component subject matter understandings).

- Invocation mapping between components (i.e. triggering behaviours in one or more components, as a result of event(s) generated in other components).

Bridges should be limited to performing these functions and should avoid implementing functionality that is defined as within the scope of a PRA component. However, since the subject matter of some PRA components encompass aspects of what are considered to be bridging functions, some exceptions need to be taken in to consideration. This is true of Semantic Translation, Data Distribution and Information Brokerage that inherently incorporate aspects of the above bridging functions. Therefore, while in simpler cases it may be appropriate for such functions to be performed by bridges in more complex cases the need for an additional PYRAMID component may be identified.

Since the role of a bridge is to perform the necessary translations to allow components with distinct subject matters to interact, they necessarily have some knowledge of the subject matter of more than one component. By using bridges, the functionality to perform such translations is isolated in to something that can be more easily managed and modified, minimising the impact of change. Bridges should generally be kept as simple as possible, but not at the expense of polluting components, i.e. they allow components to be defined in their own language and to remain agnostic of other component subject matters. Conversely, too much complexity in a bridge may indicate that it is incorrectly fulfilling aspects of the responsibilities of a PRA component as described above. This does not mean that complex bridges cannot exist in a deployment and Exploiter’s must determine whether the bridge conforms to the rules and guidance for component connections as defined in the PYRAMID Technical Standard, Ref. [1] section How to Comply with the PYRAMID Technical Standard and the PYRAMID Technical Standard Guidance document, Appendix E: Compliance Guide.

Bridges are a means of implementing counterparting, described in the Counterparting section. The three functions above are expanded upon in more detail in the Service Relationships and Counterpart Data Relationships sections, and must be considered and defined specifically for each system design to ensure successful component integration.

The following sub-sections illustrate how a bridge can connect the services, and other associated component elements, between two components, and provide an example of this.

A.3.1.4.1.1 Service Relationships

Figure 106: Service Relationships illustrates how bridges enable service relationships. It shows how a component’s service connections may connect to another component's service.

In this view, components have the following elements (note that these are instances - a component is likely to have several of each):

- Service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description. Provided_Service defines this mechanism on the component providing the capability, whereas Consumed_Service defines this mechanism on the component that needs to gain access to the capability. For more information, see Ref. [8].
- Activity is a functional operation or behaviour carried out by a component, which may invoke a functional operation or behaviour carried out by another component.
- Interface is a point of interaction defined in data terms that allows a component to interact with other components or actors.

- Attribute is an element of information that forms part of an Interface.

Additional elements that are properties of the Bridge are:

- Event_Relationship is a mapping between the timing of behaviours performed by different components.
- Interface_Relationship is a mapping between an interface on a consumed service that needs access to a capability, and an interface on a provided service that can provide this capability.
- Attribute_Relationship is a mapping between individual Attributes.

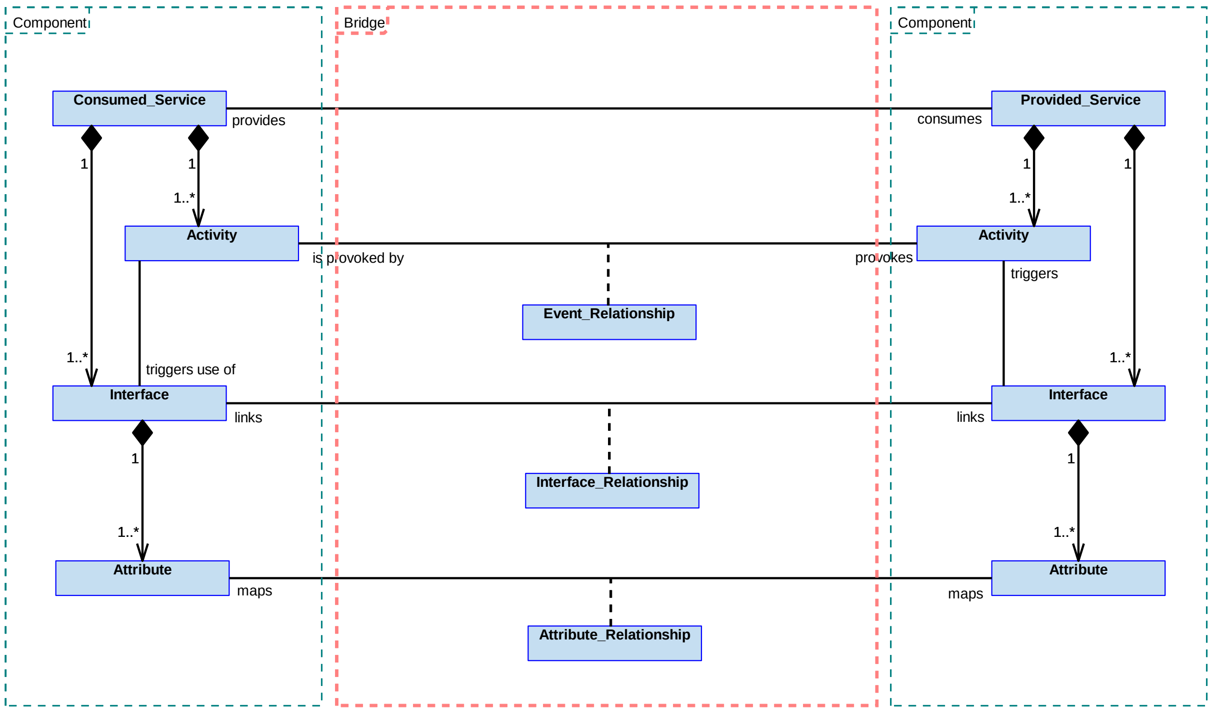


Figure 106: Service Relationships

A.3.1.4.1.2 Example of Bridge Implementation

Figure 107: Bridge Implementation Example illustrates a simple bridging example using PRA component services between the Effectors component and the Power component. In the example, the Effectors component requests that an effector is supplied with power. Note that for simplicity, not all attributes of the selected interfaces have been shown.

The bridge provides the following functionality in line with the three bridge functions:

- Data element mapping - the bridge maps the Effector_Resourcing_Request attribute of requesting_context to the Power_Delivery_Requirement attribute of sink, understanding that the context information that identifies the effector in need of a power resource can be translated into a power sink in the subject matter understanding of the Power component. It also maps the Effector_Resourcing_Request attribute of usage_profile to the Power_Delivery_Requirement attribute of power_specification.
- Data type conversion - in addition to mapping between attributes, the bridge must also convert between the data types used by the Effector_Resourcing_Request interface and the Power_Delivery_Requirement interface. In the case of usage_profile and power_specification,

this may involve converting measurements into the correct units, for example from Watts to kilowatts.

- Triggering activities in a component, driven by event(s) generated in other components - through the connectivity provided by the bridge and the interfaces, the identify_effector_resourcing_request_to_be_fulfilled activity in the Effectors component triggers the provide_power activity in the Power component.

Not all attributes of an interface need to be mapped to attributes on the corresponding interface, and attributes on one side of the bridge can be mapped to more than one attribute on the other side.

Attributes can also be used by the bridge - for example the Effector_Resourcing_Request attribute of resource is used by the bridge to determine that the type of resource being requested is power, and that the request should therefore be mapped to the Power_Delivery_Requirement interface of the Power component. If the request is for another resource type, the bridge between the Effectors and Power components can ignore the request.

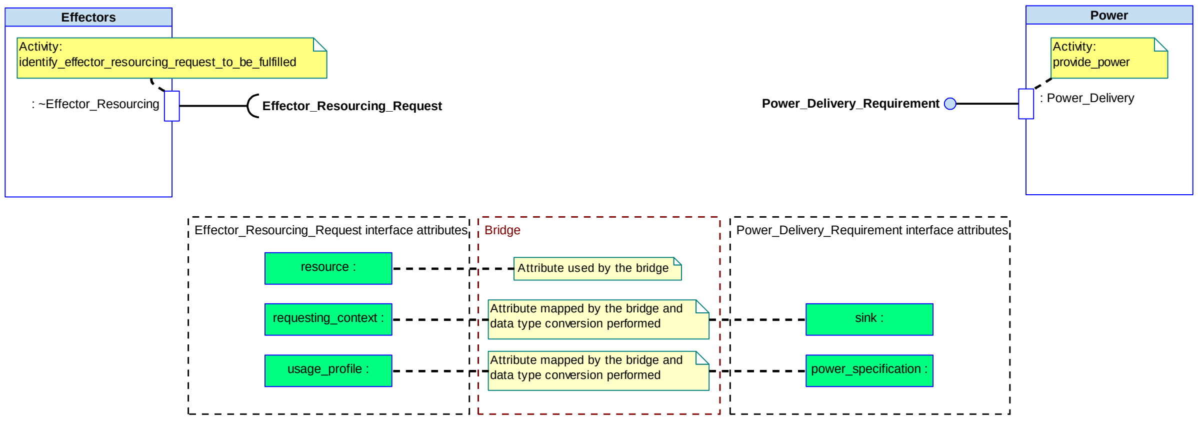

Effectors
Activity:
identify_effector_resourcing_request_to_be_fulfilled
: ~Effector_Resourcing
Effector_Resourcing_Request
Power_Delivery_Requirement
Power
Activity:
provide_power
: Power_Delivery
Effector_Resourcing_Request interface attributes
resource :
requesting_context :
usage_profile :
Bridge
Attribute used by the bridge
Attribute mapped by the bridge and data type conversion performed
Attribute mapped by the bridge and data type conversion performed
Power_Delivery_Requirement interface attributes
sink :
power_specification :

Figure 107: Bridge Implementation Example

A.3.1.4.2 Counterparting

A.3.1.4.2.1 What is Counterparting?

Counterparting is a concept whereby the same real-world object or concept is described from different viewpoints within a system dependent on the context of those viewpoints. In PRA terms, the concept can be thought of as the same real-world object or concept being described by the PRA components that reason about it in terms of their own (unique) subject matter.

The abstraction of the real-world object or concept within a component is called a counterpart. Understanding that different components contain counterparts that describe the same object or concept allows the counterparts in each component to be linked together through a counterpart relationship. At a system level, the counterparts may be thought of as being the same thing, but the components consider and act upon them differently.

For example, a weapon loaded onto an aircraft wing is understood in multiple ways by different components within a deployment:

- Inventory sees it as a physical item of inventory with characteristics that allow it to be validated as part of an inventory configuration.

- Mass and Balance sees it as an item of mass at a specific position on an aircraft.
- Target Engagement sees it as a resource that can be used to engage a target.
- Stores Release sees it as a store that can be operationally released.
- Destructive Effects sees it as a mechanism for providing destructive capability.
- Semantic Translation sees it as an external system type.

Counterparts are not limited to abstracting a singular object or concept, but can also be applied in a 'compound' sense, for example to abstract a stores release package, which contains a number of separate stores. However, formation of a counterpart relationship is only valid when both counterparts in the relationship utilise the same concept; a valid counterpart relationship cannot exist if one counterpart is abstracting a singular store and the other is abstracting the wider release package it is contained within.

Counterparting can be split into two different categories, relating to the point in the design lifecycle at which they are considered. Within this PYRAMID concept, for convenience, these are referred to as follows, despite these not being recognised terms more generally:

- **Type counterparting** - Type counterparts relate to the type of a real world object or concept (e.g. a Sidewinder missile). A type counterpart can be represented as a class in object oriented design.
- **Instance counterparting** - Instance counterparts relate to a specific instance of a real world object or concept (e.g. the specific Sidewinder missile attached to the left wingtip of an aircraft). An instance counterpart can be represented as an object in object oriented design. Instance counterparts and relationships are created, maintained, and deleted during software or model execution.

An instance counterpart is instantiated from a single type counterpart; however, a type counterpart can instantiate multiple instance counterparts:

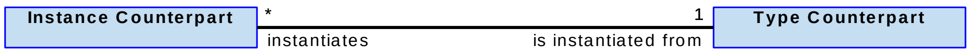


Figure 108: Type and Instance Counterparts

Access to the information within a component (including any counterparts) is via the component's interfaces. As described in the Bridges section above, interfaces on different components are connected via bridges to allow components to communicate with one another in a way that supports the separation of the components' subject matter, by maintaining loose coupling and the use of counterpart relationships to share information about real-world objects and concepts. Hence counterpart relationships between components are implemented via the bridges that connect interfaces together - commonly, the relationship will be between a provided service and a consumed service, but a provided-to-provided relationship is also possible, see the Service Relationships section. Where a counterpart relationship exists, understanding of the relationship will inform the bridge design and the required mapping of the attributes between interfaces by the bridge. This is described in more detail in Figure 107: Bridge Implementation Example.

A.3.1.4.2.2 Lifecycle of Counterpart Relationships

The lifecycle of a counterpart relationship is different for the two categories of counterparting described in this PYRAMID concept, as they represent different concepts of counterpart use (specific real world instances as opposed to classes of entity, as described in the following sub-sections).

A.3.1.4.2.2.1 Type Counterparting

Unlike instance counterparts and their relationships, type counterparts and their relationships are almost exclusively created during system design, and so there is generally no counterparting lifecycle to speak of. However, system variations in configuration may be catered for as part of system setup/initialisation or potentially even during system operation.

The design of the system may, therefore, allow for data driving options that enable the system configuration to be expressed through different data files for different overall configurations, such as different role fit equipment options. These data files may incorporate settings that define:

- Which counterpart relationships are needed, based upon which equipment and, therefore, which components are present.
- Which counterparts, within a component, need to be enabled in order to support other parts of the system (e.g. a counterpart relating to equipment that isn’t fitted may not need to be enabled).

The settings may also refine the details of a counterpart relationship or what a counterpart needs to support.

These options would normally be selected by the user, such as when planning a mission on a mission planning system, and uploaded to the system being configured.

Since type counterparts and their relationships are almost exclusively created during system design, it is worth recognising that counterparting concepts are predominately independent from the execution platform, and so are most closely associated with a platform independent design. This is due to counterparts being abstractions of real world concepts or objects. However, like most other aspects of component connections, execution platform considerations will be essential to realising successful data exchange.

A.3.1.4.2.2.2 Instance Counterparting

Just like an object within object oriented design, instance counterparts have a lifecycle during software or model execution potentially involving creation, maintenance, and deletion; therefore, so do their associated counterpart relationships.

It may be noted that for instance counterparting:

- Counterparts may exist when there is no counterpart relationship, i.e. prior to the creation of a counterpart relationship or after the counterparts have been removed from the relationship.
- The relationship can only begin to be created by the bridge when there is at least one counterpart to trigger the process.
- A counterpart relationship cannot exist with less than two related counterparts except during the creation or deletion process.

The following subsections describe simplified example scenarios involving the creation, maintenance, and deletion of instance counterpart relationships, within which the creation and deletion of instance counterparts is also described. The following should be noted when reading the example scenarios:

- Whilst they focus on counterpart relationships supporting information exchange between counterparts, other forms of exchange between counterparts are equally applicable, such as a request to provide a service to 'do work'.
- They show how no single component or bridge necessarily needs to be explicitly instructed to change the counterparting lifecycle (although in practice they could be if it is beneficial to do so); instead this is emergent behaviour.
- They focus on the counterpart relationship creation, maintenance, and deletion functions of bridges and not on other functions that will also need to be performed by bridges, such as translation of information.

Each of the generalised examples are supported by a specific example using PRA components. These involve:

- The Mass and Balance component, which contains counterparts for 'mass items'.
- The Inventory component, which contains counterparts for 'fitted stores'.
- The Stores Release component, which also contains counterparts for 'releasable stores'. Whilst releasable stores may sound like a subset of the fitted stores, they have different features or attributes.

Specific instances of these counterparts could relate to a missile fitted to an air vehicle.

A.3.1.4.2.2.2.1 Counterpart Relationship Creation Example

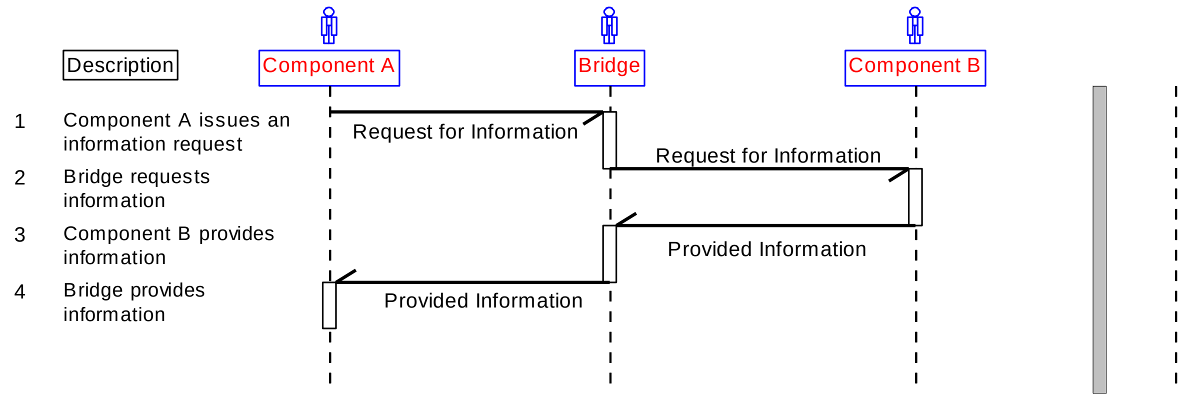

Description
Component A
Bridge
Component B
1 Component A issues an information request
2 Bridge requests information
3 Component B provides information
4 Bridge provides information
Request for Information
Request for Information
Provided Information
Provided Information

Figure 109: Counterpart Creation Example

Component A has an existing counterpart, which needs information to keep it up to date, or is in the process of creating a counterpart, which needs information to complete its creation. It issues an information request, received by the bridge, which will also trigger the process of creating a counterpart relationship. The bridge requests the information from Component B, triggering the component to generate its own counterpart (if it has not already done so) so that it can supply the information. The information is then provided to the bridge, which now recognises there are two counterparts related to the same real world object or concept, and so forms a counterpart relationship, as well as providing the information to Component A.

PRA Example:

The Mass and Balance component needs to create counterparts for all of the different variable mass items on the air vehicle (including those relating to removable stores). It issues the information request to be informed about the presence, location, and relevant mass details of any variable mass items that are fitted to the air vehicle, which is received by the bridge.

The bridge recognises that different mass items have different counterpart relationships involving different components and so will manage all of the different relationships; for example, items relating to fuel have different counterpart relationships to those relating to stores. However, this example focuses on the counterpart relationships with the stores.

The bridge requests the relevant fitted store information from the Inventory component, which already has counterparts relating to each store that is present on the aircraft. Inventory provides the requested information to the bridge, allowing it to complete the different counterpart relationships. The bridge provides the information to Mass and Balance, allowing it to complete the creation of its mass item counterparts, noting that the bridge may translate the store type (provided by Inventory) to a mass value (provided to Mass and Balance) and the store station (provided by Inventory) to an aircraft location (provided to Mass and Balance).

A.3.1.4.2.2.2.2 Counterpart Relationship Maintenance Example

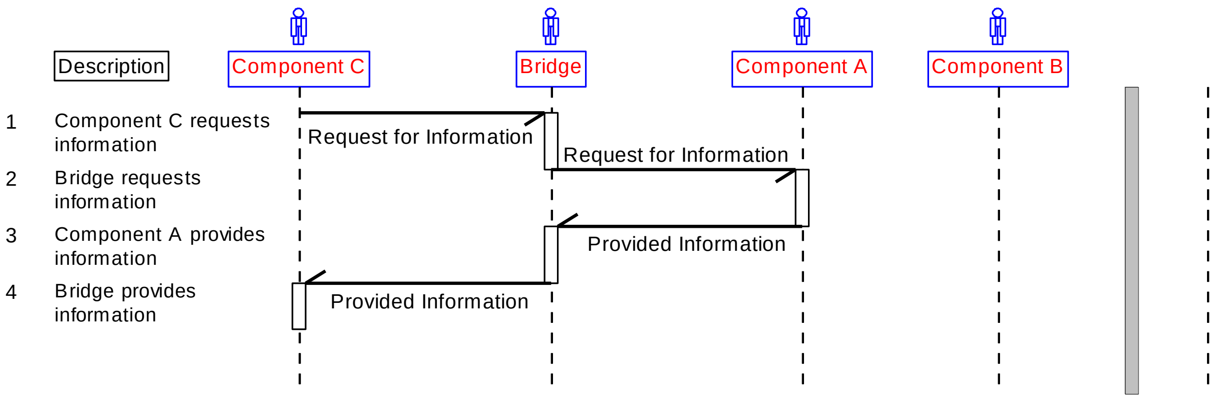


Figure 110: Counterpart Maintenance Example

This example follows on from the creation example, where counterparts in components A and B already have an established counterpart relationship.

Component C needs to create a counterpart, which needs information to complete the creation process. It issues an information request, which is received by the bridge. The bridge recognises that a relevant counterpart relationship is already present and so adds the counterpart in Component C to the counterpart relationship. The bridge requests the information from Component A, which provides the requested information. The bridge provides the information to Component C, which uses it to complete the creation of its counterpart.

PRA Example:

The Inventory and Mass and Balance components already have counterparts related through counterpart relationships (as described in the previous, counterpart creation example).

The Stores Release component has been given a requirement to release a missile of a specified type. It therefore needs to create counterparts for all of the different stores and so needs to know which releasable store types are fitted and on which store stations. (Note that it is assumed that the presence of other releasable store types, not being released, may influence which missile is selected for release.)

This information request is, via the bridge, placed on the Inventory component, which already has counterparts relating to each store - these contain information about the station that the store is located on. Inventory provides the requested information, which is received by the bridge. The bridge provides the information to Stores Release, which is able to complete the creation of its counterparts. The bridge also uses this information to update the existing counterpart relationships involving Inventory and Mass and Balance to also include Stores Release. Doing so not only supports information exchange between Inventory and Stores Release, but also allows Mass and Balance to be informed about any planned release of a mass item, sourced from Stores Release, to support planning the redistribution of mass accordingly.

A.3.1.4.2.2.2.3 Counterpart Relationship Deletion Example

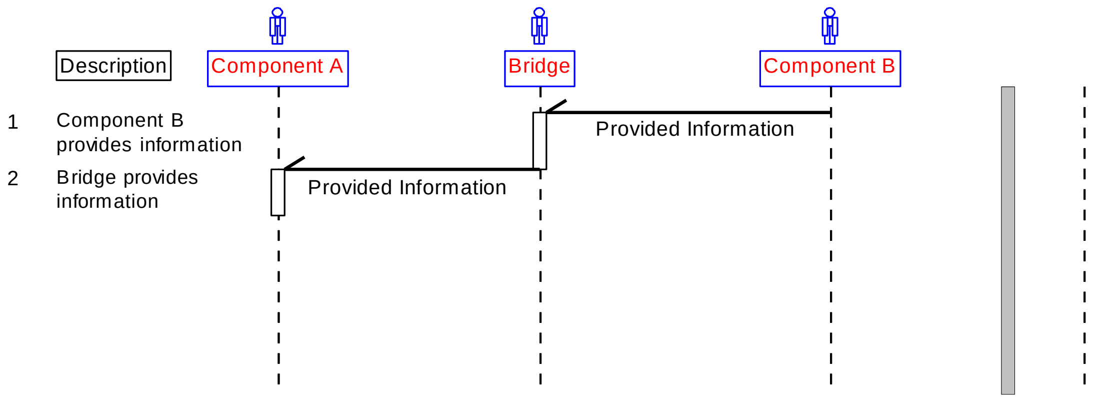


Figure 111: Counterpart Deletion Example

There may come a time when a counterpart or a counterpart relationship is no longer useful and may need to be deleted. However, careful consideration should be given to the sequencing of deletion events, in order to maintain system consistency, and maintaining records of what has happened.

Component B provides information relating to its counterpart, which the bridge provides to Component A. The information indicates to Component A that its counterpart is no longer relevant and so triggers its deletion. The bridge also recognises that the information implies that there will no longer be a meaningful counterpart relationship between the counterparts in the two components; therefore, having confirmed receipt of the information by Component A, the bridge deletes the counterpart relationship.

Deletion of a counterpart relationship may also occur under other circumstances, such as if the bridge recognises that a component is no longer functional in a way needed to support the counterpart relationship (e.g. if a component permanently loses communication with the bridge).

PRA Example:

Following the release of a store, the Inventory component provides information indicating the new state of the store on the store station. Based on the existing counterpart relationships, the bridge informs Mass and Balance that the mass item is no longer fitted. Mass and Balance uses this information and may use it to delete the counterpart if it is no longer needed. In this case, following acknowledgment that Mass and Balance has received the information, the bridge deletes the counterpart relationship since it is no longer needed.

A.3.1.4.2.3 Counterpart Data Relationships

This sub-section illustrates how a counterpart relationship is supported by passing data between components via interfaces. This data relationship is applicable to both type counterparts and instance counterparts.

Figure 112: Data Modelling Relationships Diagram shows two components with a counterpart relationship expressed by a bridge.

The Conceptual_Thing is a real-world object or concept of interest to the system about which information can be known. Such an entity may or may not exist in the physical world - for example, a vehicle that has been detected, or a potential threat that may not materialise. This same Conceptual_Thing may be represented by different data in multiple components, in accordance with their differing roles. This is a design concept, which manifests itself as an element of the component connection (bridging) design.

Other elements shown in the diagram are described and related as follows:

- Component is a reusable configurable unit with knowledge of a specific Specific_Subject_Matter area.
- Specific_Subject_Matter is the component specific viewpoint of the information about Conceptual_Things.
- Counterparted_Item is an information entity (e.g. a software class or object) defined as part of the internal structure of a component that has a counterpart relationship with a similar entity in another component due to them both describing the same real-world object or concept. A counterpart does not have to describe a single physical item - a stores release package (containing multiple stores) could be described as a counterpart from the weapon release and jettison viewpoints. However, each store within that package could also be counterparted separately from an inventory or cockpit display viewpoint.
- Interface is a point of interaction defined in data terms that allows a component to interact with other components or actors. Interfaces are the common link between this diagram and Figure 106: Service Relationships.
- Bridge is an interconnection enabling a service required by a component to be satisfied by defining how that service maps to other services to satisfy its needs.
- Data_Conversion is the process of translating between data formats used by different components. This process is carried out inside a Bridge.
- Semantic_Conversion mechanises the process of translating the meaning of data from one Specific_Subject_Matter area into another. This is another function of the Bridge.
- Interface_Relationship is a connection between Counterparted_Items in different Components that defines how those counterparts relate to one another. An interface relationship is

implemented in software (e.g. as a class association or object link) through component interfaces via the use of a bridge in order to support loose coupling between the components.

- Counterpart_Relationship is the relationship between Counterparted_Items held by different components represented on their respective Interfaces. This mapping is configured in a Bridge, which brings together both Data_Conversion and Semantic_Conversion. For example, converting the quantity of fuel (in litres) that is being used to propel an aircraft into the mass of the fuel (in kilograms), which is relevant to mass and balance calculations.

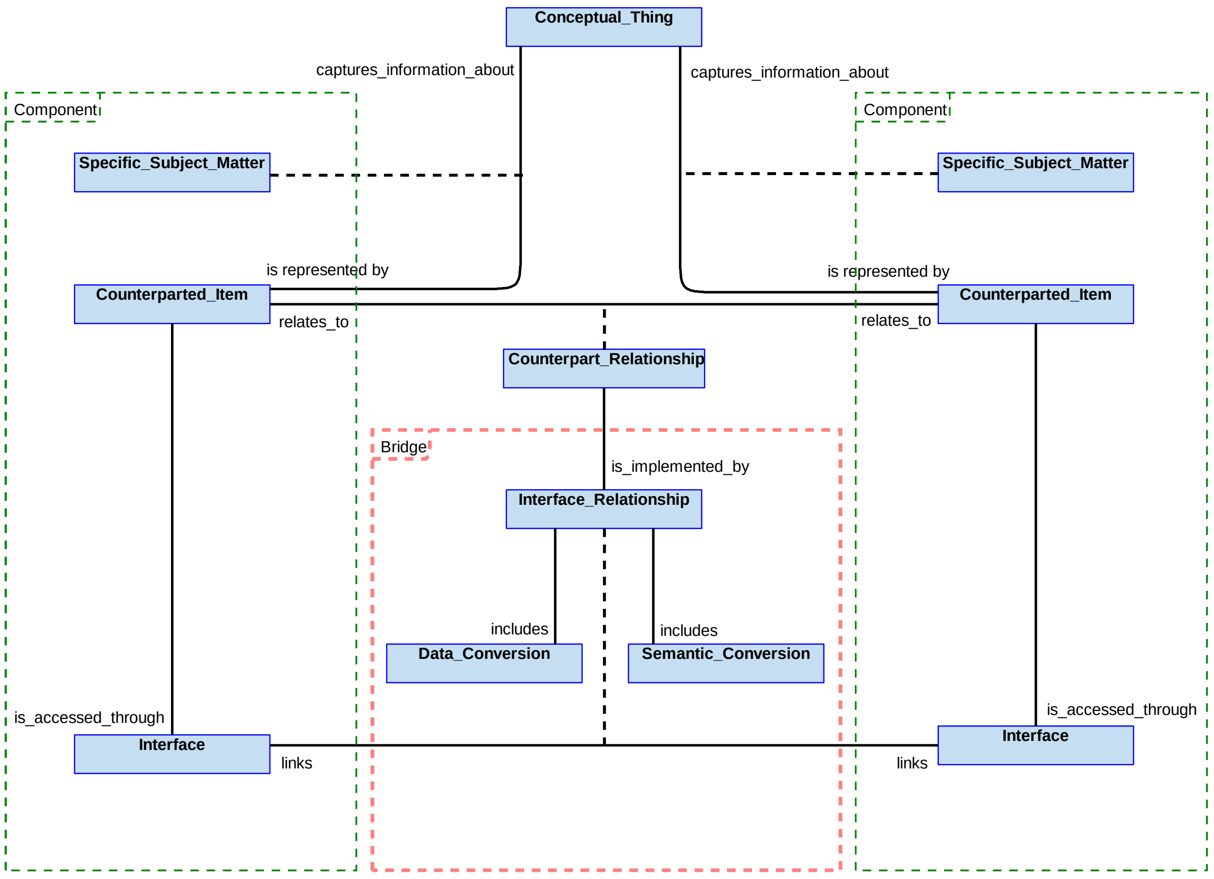


Figure 112: Data Modelling Relationships Diagram

A.3.1.5 Component Interactions

Understanding of the information relationships between components will allow the services on different components (through their interfaces and connecting bridges) to be linked together. However, depending on the complexity of the interactions between components that will provide the functionality of the working system, those interactions may need to be further defined in order to ensure the expected system behaviour is achieved.

For cases where these interactions are not complex, see Simple Transactions below.

For cases where the interactions are complex, a number of methods can be used to define them, including:

- The use of service contracts
- The use of component interaction patterns

A.3.1.5.1 Simple Transactions

The most straightforward of component connections are simple transactions, where one component interacts with another component using a single transaction between a service on one component and a service on the other (with or without a response). This can be used to pass control from one component to the other, or to request or provide information. Simple transactions do not require any detailed representation, with the connection often implied from the two service definitions.

There may still be some complexity in the implementation introduced by the chosen messaging protocol. The messaging protocol can have a drastic impact on the service presentation. For example, the same core logic may use a remote procedure call in one deployment and a publish-subscribe process in another. Deciding which connection protocol is appropriate will depend on the deployment.

A.3.1.5.2 Service Contracts

A.3.1.5.2.1 What is a Service Contract?

Service contracts are used to define relationships between services in order to provide a capability. This allows the components that provide these services to work together in order to complete the task expressed in the service contract.

The full service contract shows both the provided services and consumed services that are used. This includes mandatory services that are required to complete the task along with any optional services. Optional services can be used to allow alternative workflows or to provide supplementary information relating to the activities being performed.

The relationships between the provided services and consumed services can be shown on a service contract diagram, see Figure 113: Service Contract Diagram.

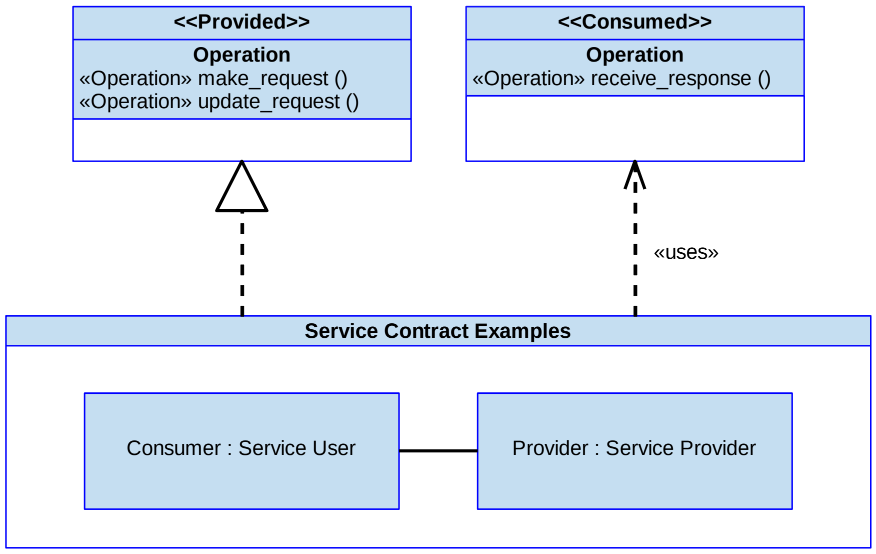

<<Provided>>
Operation
«Operation» make_request ()
«Operation» update_request ()
<<Consumed>>
Operation
«Operation» receive_response ()
«uses»
Service Contract Examples
Consumer : Service User
Provider : Service Provider

Figure 113: Service Contract Diagram

The provided services are shown as a realisation of what the provider provides, while the consumed services are shown as usage by the consumer.

When defining attributes and behaviour of the services, it is important to remember that the service object only exists for the life of the service. It will remove all information local to the service contract after it is complete. If necessary, additional services can be used to pass this data out to a different attribute or operation outside of the service contract.

The relationships between the provided services and consumed services can be shown on a choreography diagram, see Figure 114: Service Contract Sequence.

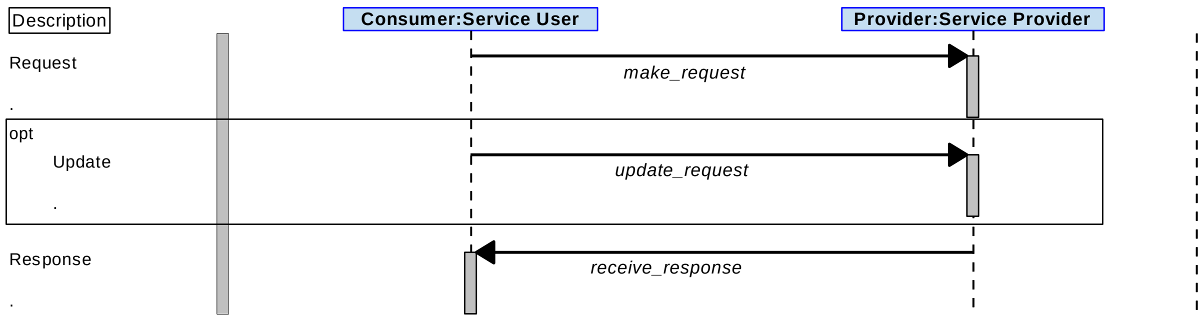


Figure 114: Service Contract Sequence

A service contract can also be defined from the perspective of only one of the components involved, providing a component centric view. This specifies the required pattern of use of either a consumed service or a provided service, where a sequence or order is essential to the correct operation of the service.

Figure 115: Component Centric Service Contract Sequence shows an example service contract sequence diagram for the Vehicle Guidance component service, Trajectory_Demand. This example is to illustrate the use of a service contract; it should not be interpreted as prescriptive of the structure of a Vehicle Guidance service.

The Consumer in this case is shown as a generic component needing to place a demand for the movement of the air vehicle. The diagram captures the sequence of interactions that, in this example, must be undertaken in the defined order. Other examples might define unordered or partially ordered steps. In this respect, the service contract is specifying the flexibility that is required of the provided service.

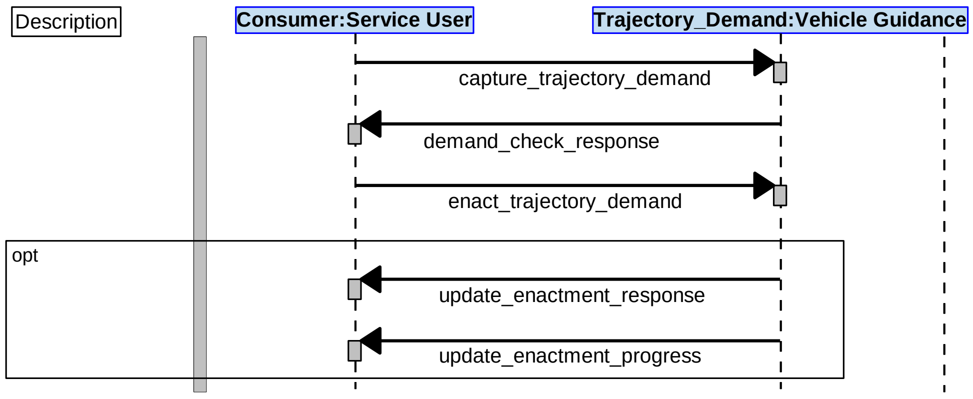

Description
Consumer:Service User
Trajectory_Demand:Vehicle Guidance
capture_trajectory_demand
demand_check_response
enact_trajectory_demand
opt
update_enactment_response
update_enactment_progress

Figure 115: Component Centric Service Contract Sequence

In this example, the Consumer places a requirement to move the air vehicle. Vehicle Guidance provides an ‘in principle’ confirmation that the demand can be enacted as it is not currently inhibited by any constraints or conflicting demands. At a subsequent time, the Consumer requests the enactment of the demand. Vehicle Guidance confirms enactment and provides subsequent progress updates (e.g. demand completed or demand interrupted). The enactment response and enactment progress interactions are shown as optional in that a Consumer need not necessarily support these interactions.

For more information on service contract diagrams and choreography diagrams, refer to the SoaML Standard, Ref. [19].

A.3.1.5.2.2 Whether to Use a Service Contract

Service contracts are useful in defining complex, multi-component service interactions. If a simple transaction can perform the required information exchange, a service contract is not needed.

When adding services to the service contract, it is important to include only the services relevant to the contract. There may be other actions, which while related, can be carried out independently of the service contract.

The duration or frequency at which a service is requested should also be considered. Even for a small number of interacting services, if the system needs to support concurrent processing of multiple service requests, having a well-defined service contract will allow the system to scale more easily to the requests that are made of it.

A.3.1.5.3 Component Interaction Patterns

The collaborative behaviour of the services can be further refined by applying a component interaction pattern.

The interaction pattern provides additional rules that govern the operation of the services; it is not a service pattern itself (these are found in the PYRAMID Technical Standard, Ref. [1], section Component Composition, sub-section Services). The logic for controlling the sequence of activities should still be managed by the components. Depending on the type of pattern that is needed, this may mean one component is in control of the others, or it could be that a set of rules are provided indicating how components should communicate their progress.

Component interaction patterns can have the concept of activation. An Activation service detailed in the pattern can be called from an external source known as the Activator. This can be used to provide information into the service contract relating to the service request. Upon activation, the Participant components are selected to complete the service request.

A component interaction pattern may define a specific role for a controlling component (e.g. Coordinator or Orchestrator). The common terms Participant and Subordinate are used across the different patterns. Participants are components that provide or consume services that are required by the service contract. Subordinates are the components that are needed to facilitate the completion of the services required by the pattern.

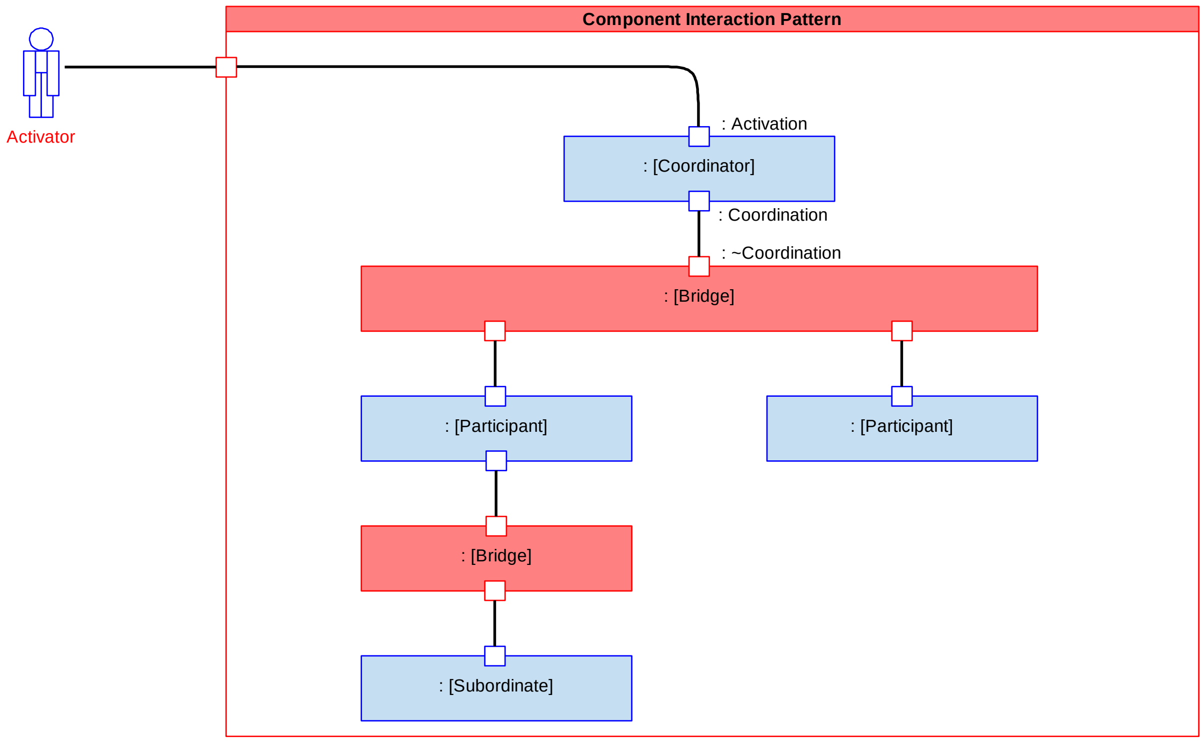


Figure 116: Component Interaction Pattern Roles

Where differing mechanisms of coordination are needed to complete a task, it may be necessary to link multiple patterns together. This is called a compound pattern. Each section of a compound pattern can be treated as if it was one of the individual component interaction patterns.

When defining the pattern, it should:

- Specify the mandatory services that are provided or consumed.
- Identify the activation service (where applicable).

Additionally, it may:

- Identify optional services.
- Specify quality of service measurements.
- Place constraints on the usage of the services.

The job of a system integrator could become very complex if every component connection behaved in a unique way. The solution is to produce a standard set of interaction behaviour types.

Simpler transactions are easier to develop and maintain. Using subject matter based components within the PRA reduces the number of connections between components, allowing for fewer component connections. However, one of the side effects of this approach is that there is more likely to be a need to coordinate behaviour between components.

The counterparts allow information to be linked between components, but to coordinate behaviour different patterns are needed. Four common patterns to manage this are singled out by Thomas Erl for special consideration in Ref. [7]. They are summarised below:

- **Atomic Transactions** are the most common coordination type. They are used when actions or data need to be synchronised between components.
- **Business Agreement** is used when the priorities of multiple parts of the system need to be coordinated when coming up with a solution. Requirements are derived by the coordinating component, which places requirements on participating components, often including criteria for measuring success. How a solution is determined and enacted is left up to the participating components.
- **Orchestration** is similar to a Business Agreement, except the orchestrating component owns the process, setting up other components to do different parts or steps of the process.
- **Choreography** is a different type of collaborative behaviour, where partner components contact a choreography service to discover the rules and process for working with other partner components. This is particularly useful in an extended system with numerous peer components across multiple platforms.

Decisions about choices between using a component interaction pattern and using simple transactions will depend on the Exploiting Programme.

A.3.1.5.3.1 Examples of Component Interaction Patterns

A.3.1.5.3.1.1 Coordination – Atomic Transaction

One of the most commonly used component interaction patterns is atomic transaction coordination. It is used when activities or data need to be synchronised between components. For example, atomic transaction coordination can be used for managed resource allocation, when moving to new modes of operation, or for data loading.

This service pattern is only appropriate if the transaction passes the ACID test, which means that it is:

- **Atomic** - All participants, including any subordinate components, can be treated as a single 'atomic' unit, with all components changing together or all failing together.
- **Consistent** - None of the changes violate the validity of data in the components; if they do all the changes need to be rolled back.
- **Isolated** - Multiple instances of the service can occur concurrently, without interfering with each other.
- **Durable** - Upon successful completion the change will survive any subsequent failures of new transactions.

There are several protocol definitions for this service pattern, and other similar ACID transactions; one example is the Web Service Atomic Transaction standard, Ref. [42].

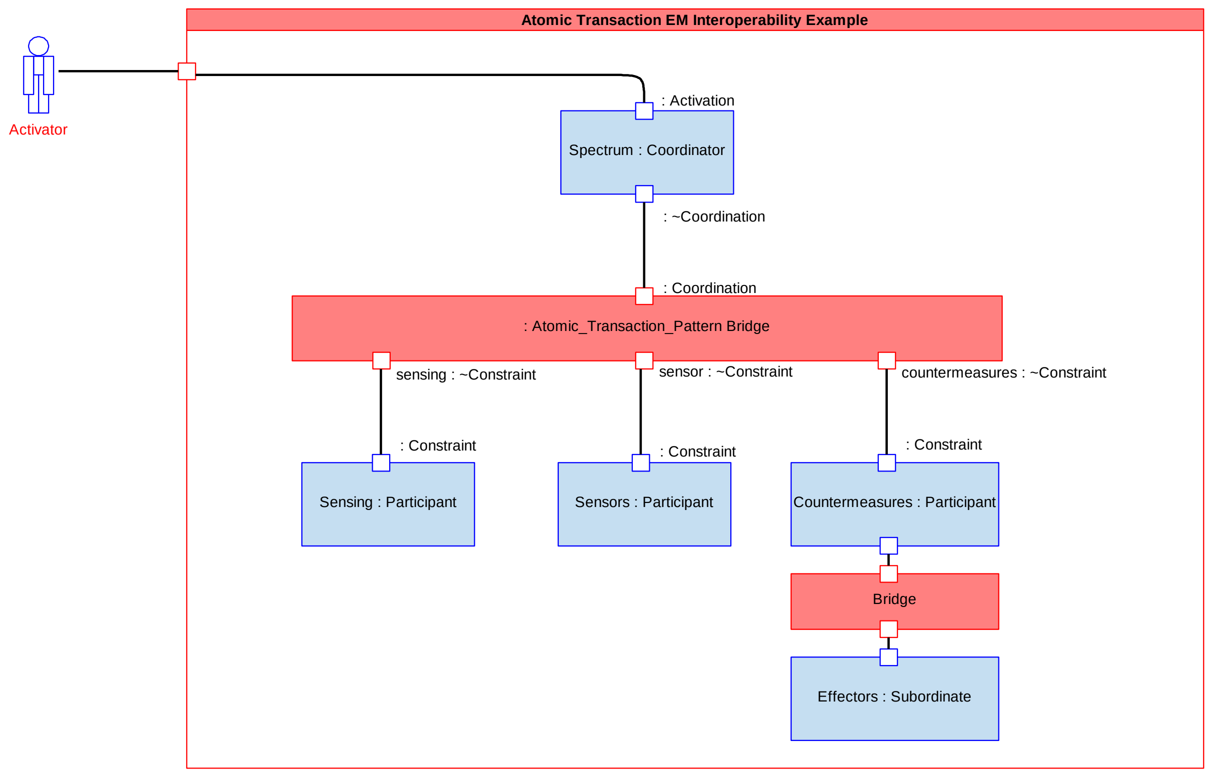
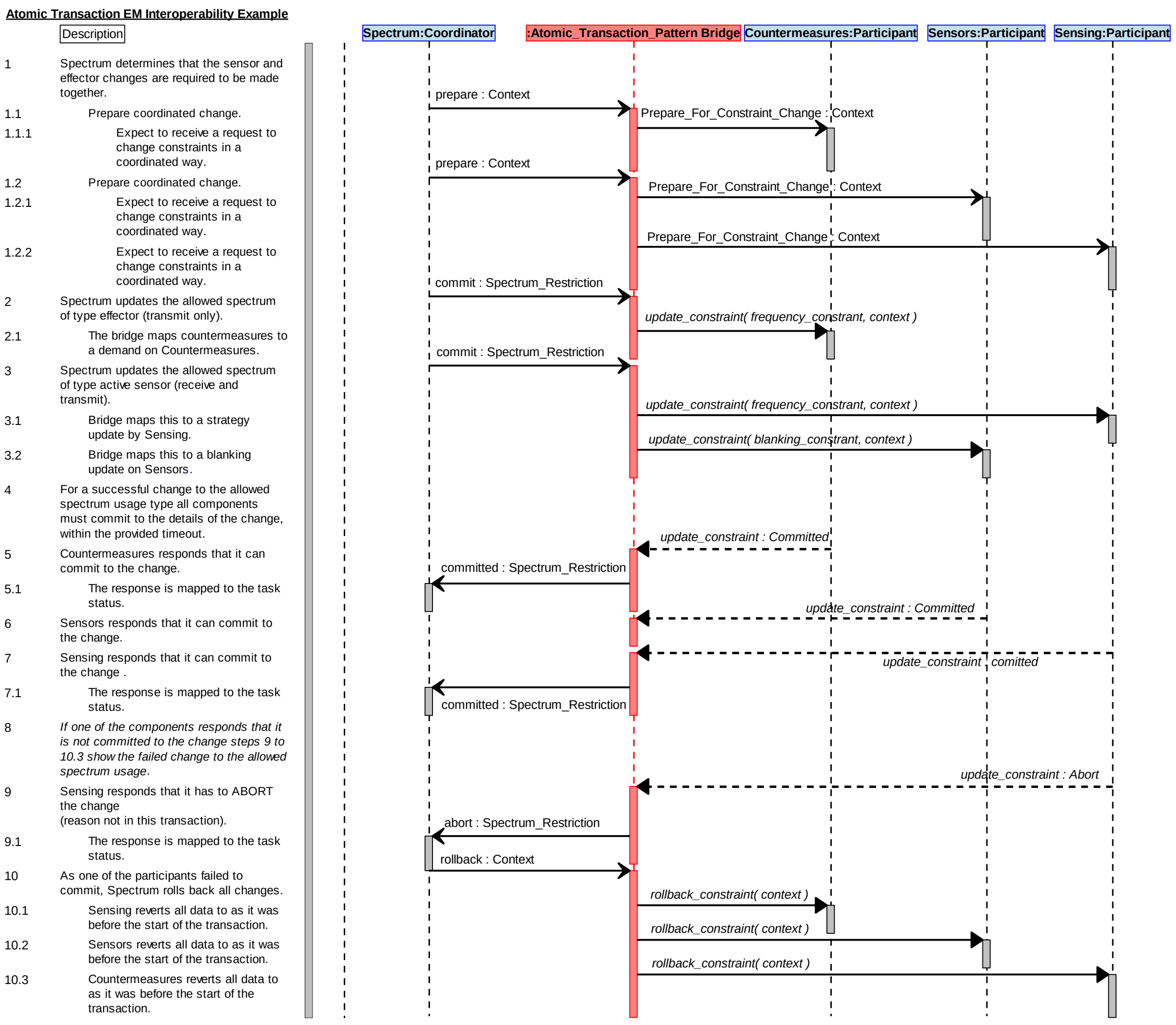


Figure 117: Atomic Transaction EM Interoperability

The details of an atomic transaction for EM operability can be seen in Figure 118: Atomic Transaction EM Interoperability Sequence Diagram.

In this example three components (Sensing, Sensors and Countermeasures) have their spectrum usage restricted (i.e. transmitting, receiving or both) in a coordinated way by the Spectrum component. The spectrum use restriction is added as a change to the constraint on all of the involved components. To avoid any conflicting frequency use all components need to be updated together.



Atomic Transaction EM Interoperability Example

	Description
1	Spectrum determines that the sensor and effector changes are required to be made together.
1.1	Prepare coordinated change.
1.1.1	Expect to receive a request to change constraints in a coordinated way.
1.2	Prepare coordinated change.
1.2.1	Expect to receive a request to change constraints in a coordinated way.
1.2.2	Expect to receive a request to change constraints in a coordinated way.
2	Spectrum updates the allowed spectrum of type effector (transmit only).
2.1	The bridge maps countermeasures to a demand on Countermeasures.
3	Spectrum updates the allowed spectrum of type active sensor (receive and transmit).
3.1	Bridge maps this to a strategy update by Sensing.
3.2	Bridge maps this to a blanking update on Sensors.
4	For a successful change to the allowed spectrum usage type all components must commit to the details of the change, within the provided timeout.
5	Countermeasures responds that it can commit to the change.
5.1	The response is mapped to the task status.
6	Sensors responds that it can commit to the change.
7	Sensing responds that it can commit to the change .
7.1	The response is mapped to the task status.
8	*If one of the components responds that it is not committed to the change steps 9 to 10.3 show the failed change to the allowed spectrum usage.*
9	Sensing responds that it has to ABORT the change (reason not in this transaction).
9.1	The response is mapped to the task status.
10	As one of the participants failed to commit, Spectrum rolls back all changes.
10.1	Sensing reverts all data to as it was before the start of the transaction.
10.2	Sensors reverts all data to as it was before the start of the transaction.
10.3	Countermeasures reverts all data to as it was before the start of the transaction.

Spectrum:Coordinator
:Atomic_Transaction_Pattern Bridge
Countermeasures:Participant
Sensors:Participant
Sensing:Participant
prepare : Context
Prepare_For_Constraint_Change : Context
prepare : Context
Prepare_For_Constraint_Change : Context
Prepare_For_Constraint_Change : Context
commit : Spectrum_Restriction
update_constraint(frequency_constrant, context)
commit : Spectrum_Restriction
update_constraint(frequency_constrant, context)
update_constraint(blanking_constrant, context)
update_constraint : Committed
committed : Spectrum_Restriction
update_constraint : Committed
update_constraint : comitted
committed : Spectrum_Restriction
update_constraint : Abort
abort : Spectrum_Restriction
rollback : Context
rollback_constraint(context)
rollback_constraint(context)
rollback_constraint(context)

Figure 118: Atomic Transaction EM Interoperability Sequence Diagram

First Spectrum prepares those involved for the coordinated change. As components are agnostic of other components, with requests based on their own understanding, it is possible that there is not always a one-to-one match between coordination requests and component services; this is managed as a mapping in the bridge. In this example the bridge duplicates the sensor request to the Sensing and Sensors components. The prepared request includes the type of change (but not the details), allowing components to also prepare subordinate components for the change. They can do this using a simple transaction or another atomic transaction (often using the same ID).

The second stage of the transaction is a request for all components to commit to the details of a change, within the provided timeout. If any of the components fail to respond within the timeout, or as in this example one of the components responds that it has to abort, then Spectrum requests a rollback from all involved, resetting all the constraints to how they were before the transaction started. The resolution to why the transaction was aborted is the subject of a different set of transactions.

In this example all the properties changed are of the same type, but that does not have to be the case, as long as the change passes the ACID test. This kind of transaction is often used even with single participants, allowing them to prepare subordinate components for the change. However the isolation requirement means that this coordination type is only really appropriate to small activities or data changes; for complex activities that require coordination, an alternative approach is required, such as the Business Agreement covered in the next section.

A.3.1.5.3.1.2 Coordination – Business Agreement

The second common component interaction pattern is Business Agreement coordination. This is used when the priorities of multiple parts of the system need to be coordinated when coming up with a solution. For example, as far as the Sensors component is concerned, the best search solution may be to turn on all active sensors, whereas this is the worst solution as far as the Observability component is concerned; a middle ground that both are happy with needs to be chosen.

The coordinator's aim is to reach an agreement on a level of service based around a set of derived requirements, with the solution owned by the participants.

As with the Atomic Transaction coordination there are several standards for this service pattern; one example is the Web Service Business Activity standard, Ref. [43].

The example Business Agreement Coordination pattern shown in Figure 119: Business Agreement Jettison Example is a snapshot of a single coordination from the Jettison Management IV, where Jettison asks Asset Transitions to determine a solution that will make the jettison package releasable (i.e. opening doors before release), and requests Stores Release to provide a stores jettison solution.

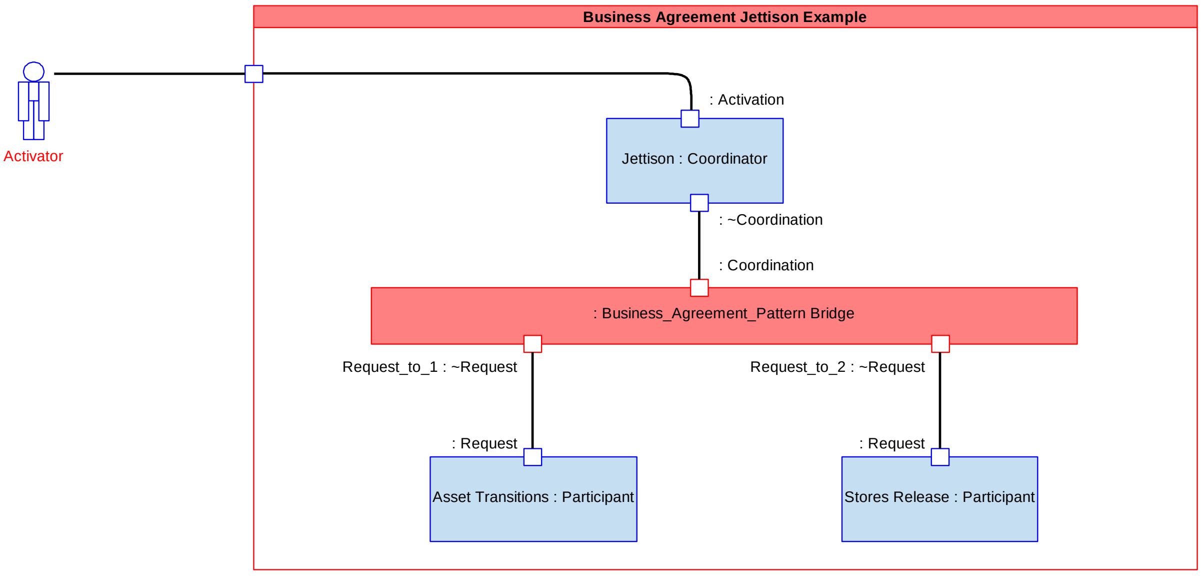


Figure 119: Business Agreement Jettison Example

As can be seen in Figure 120: Business Agreement Sequence Diagram Jettison Example, this component interaction pattern begins with requirements being placed on the two components from Jettison. The requirements take the form of one or more criteria and a measurement of success. All participants determine a proposed solution against their requirements, and provide scores against the criteria.

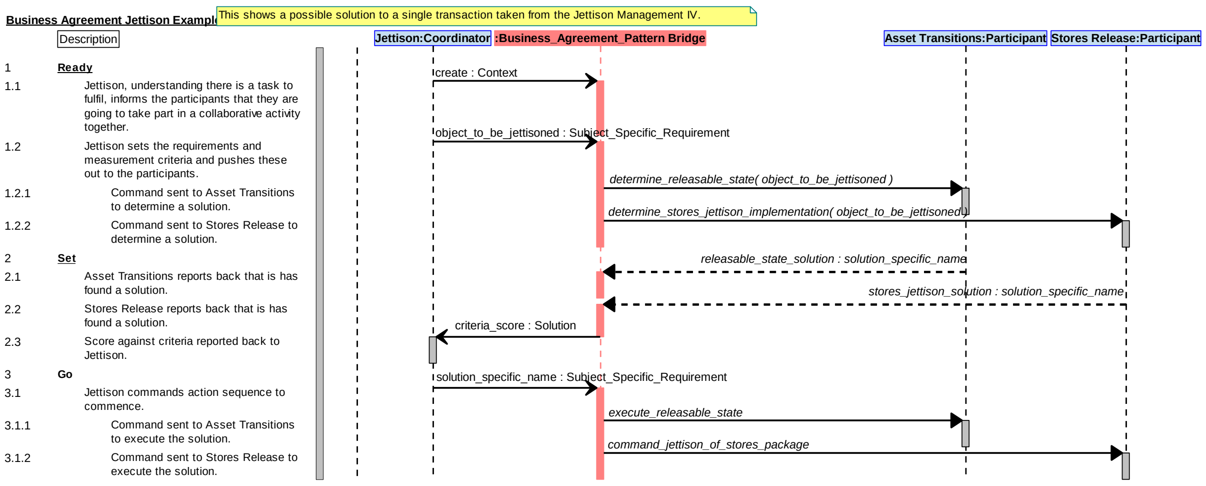

Business Agreement Jettison Example
This shows a possible solution to a single transaction taken from the Jettison Management IV.
Description
1 Ready
1.1 Jettison, understanding there is a task to fulfil, informs the participants that they are going to take part in a collaborative activity together.
1.2 Jettison sets the requirements and measurement criteria and pushes these out to the participants.
1.2.1 Command sent to Asset Transitions to determine a solution.
1.2.2 Command sent to Stores Release to determine a solution.
2 Set
2.1 Asset Transitions reports back that is has found a solution.
2.2 Stores Release reports back that is has found a solution.
2.3 Score against criteria reported back to Jettison.
3 Go
3.1 Jettison commands action sequence to commence.
3.1.1 Command sent to Asset Transitions to execute the solution.
3.1.2 Command sent to Stores Release to execute the solution.
Jettison:Coordinator
:Business_Agreement_Pattern Bridge
Asset Transitions:Participant
Stores Release:Participant
create : Context
object_to_be_jettisoned : Subject_Specific_Requirement
determine_releasable_state(object_to_be_jettisoned)
determine_stores_jettison_implementation(object_to_be_jettisoned)
releasable_state_solution : solution_specific_name
stores_jettison_solution : solution_specific_name
criteria_score : Solution
solution_specific_name : Subject_Specific_Requirement
execute_releasable_state
command_jettison_of_stores_package

Figure 120: Business Agreement Sequence Diagram Jettison Example

The second stage is the coordinator selecting the solution to enact. As the actual details of the solution are based on the subject matter of participant components (not the coordinator), the coordinator selects the best solution based on the score of the solutions and the score of any pre-requisites required for the solution. In this case the criterion provided by Jettison is binary (you can jettison or not), but for other transactions this is likely to be a more nuanced measure, for example including time or resource cost. Once a solution is selected by the coordinator, the participants are informed of the choice and when the solution is to be enacted (at an agreed time or on demand).

The final stage of the pattern is for the participants to enact the solution. Jettison can monitor progress of achievement against the requirement. Asset Transitions and Stores Release are best placed to determine whether their own requirements have been met, and the coordinator determines completion of the overall requirement.

Note: If for any reason the service level agreed against the requirement can no longer be met, then the coordinator is to be informed; however, if the requirement can still be met by a new solution without impacting other components, this is within the scope of the participants to select.

Business Agreement Coordination can often be for very long term and high level goals. This leads to a further derivation of requirements within the deployment as part of the top level requirement (as shown in Figure 121: Business Agreement Generic Hierarchical Example), with each subject matter area owning its own solution, and derived requirements.

This coordination differs from an Atomic Transaction in three key ways:

- The coordinator only owns derivation of the requirements; the participant determines how the solution is achieved, allowing it to be used at an abstract level.
- Business Agreement transactions involve requirement derivation and solution determination, followed by solution selection and then a final enactment stage, whereas the steps in an Atomic Transaction are just for the components to prepare and then commit.
- There is no requirement to be able to roll-back to a previous state, allowing it to be used for long term irreversible change such as ‘fly to this location’.

It should be noted that the pattern for solution derivation does not require 'coordination'. As in the Business Agreement Jettison Example above, if the derived requirements and solutions are independent of each other they can be treated as separate transactions. The ‘keeping track’ of the dependencies can then be considered ‘invisible’ outside of the coordinating component. Taken to its logical extreme this stops becoming so much of a coordination between participants and more a directed series of steps, and this approach is covered in the orchestration pattern in the next section.

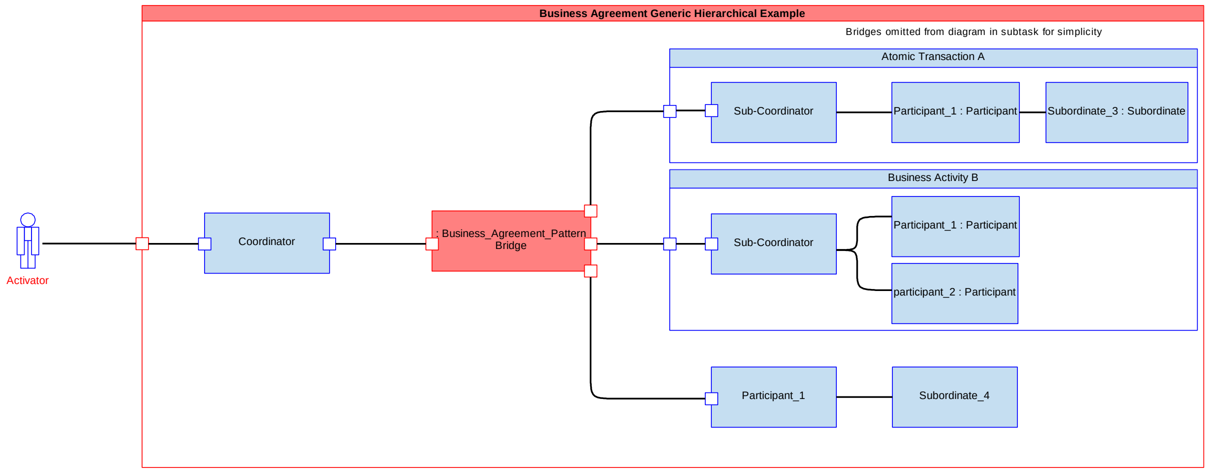

Business Agreement Generic Hierarchical Example
Bridges omitted from diagram in subtask for simplicity
Atomic Transaction A
Sub-Coordinator
Participant_1 : Participant
Subordinate_3 : Subordinate
Business Activity B
Sub-Coordinator
Participant_1 : Participant
participant_2 : Participant
Activator
Coordinator
: Business_Agreement_Pattern Bridge
Participant_1
Subordinate_4

Figure 121: Business Agreement Generic Hierarchical Example

A.3.1.5.3.1.3 Orchestration

The Orchestration component interaction pattern differs from a coordination transaction in that the orchestrator owns the workflow logic, rather than just the requirement derivation. This can be a very versatile approach, where multiple components can be reused in a variety of different configurations and orders to achieve different results.

The process itself is usually data-driven, allowing a lot of flexibility, with multiple standards existing for the process execution; one example is the Web Service Business Process Execution Language standard, Ref. [44].

The example Orchestration component interaction pattern (Figure 122: Orchestration Startup Example Diagram) shows Asset Transitions in a role that is similar to the role it plays in the Figure 176: Startup IV. In this interaction view, Asset Transitions manages the workflow of starting up an air vehicle across multiple interactions, driving out the process steps one by one to achieve an overall result. This pattern involves orchestrating multiple component interaction patterns across multiple services, either in parallel or in series depending on the workflow dependencies. This example involves three stages all activated by the same request.

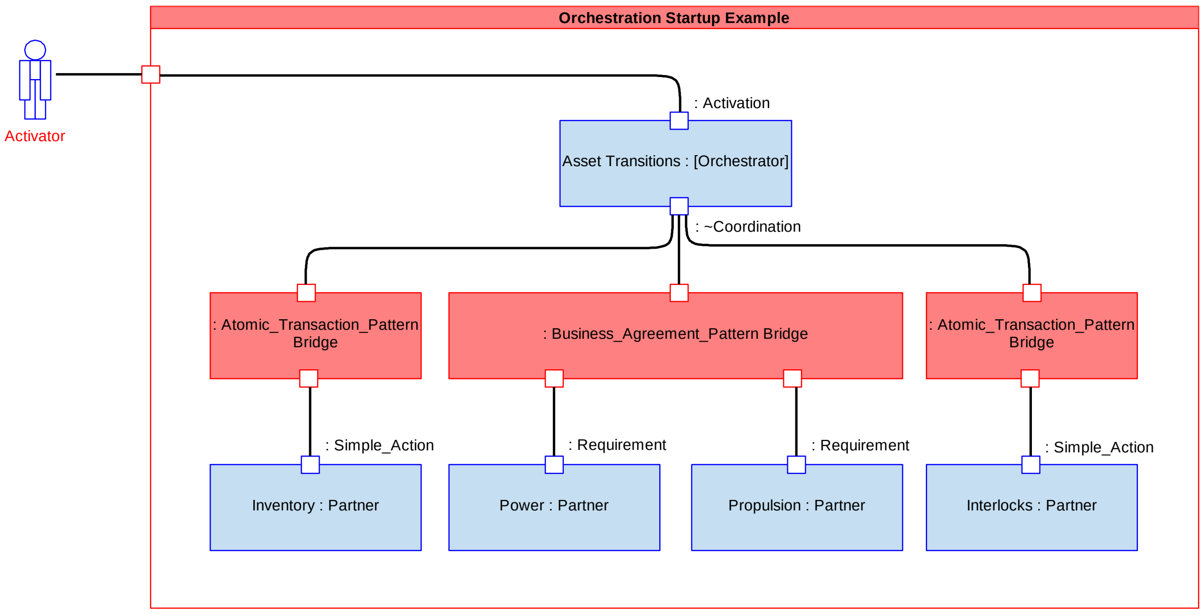

Orchestration Startup Example
Activator
: Activation
Asset Transitions : [Orchestrator]
: ~Coordination
: Atomic_Transaction_Pattern Bridge
: Business_Agreement_Pattern Bridge
: Atomic_Transaction_Pattern Bridge
: Simple_Action
: Requirement
: Requirement
: Simple_Action
Inventory : Partner
Power : Partner
Propulsion : Partner
Interlocks : Partner

Figure 122: Orchestration Startup Example Diagram

As can be seen in Figure 123: Orchestration Startup Example Sequence Diagram, this component interaction pattern begins with Asset Transitions determining the correct sequence of steps and dependencies. The first step is to check with Interlocks that the correct authorisations and Interlocks have been fulfilled to allow startup. Asset Transitions doesn't need to know the details of what is required, just whether permission is granted or not.

When this activity is complete Asset Transitions requests that Power and Propulsion both determine if they can bring basic power online at a given time. This is done as a business agreement coordination, because power and propulsion systems may be able to start up independently but not together. Again Asset Transitions does not need to know the details (they are outside its subject matter area), just whether a solution for initial power is in place. As a solution exists, application of limited power to a store attachment mechanism can be started.

The last step of the process flow shown is the triggering of Inventory to do a stores fit check. In reality there will be further checks but this illustrates the pattern.

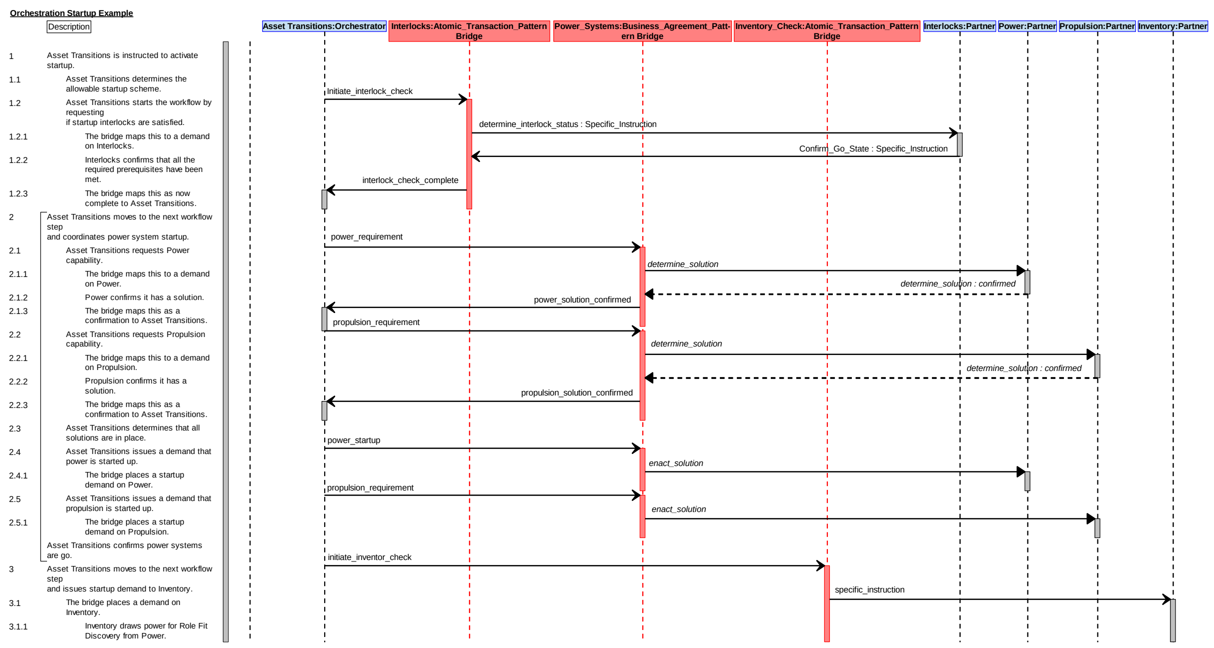

Orchestration Startup Example
Description
1 Asset Transitions is instructed to activate startup.
1.1 Asset Transitions determines the allowable startup scheme.
1.2 Asset Transitions starts the workflow by requesting if startup interlocks are satisfied.
1.2.1 The bridge maps this to a demand on Interlocks.
1.2.2 Interlocks confirms that all the required prerequisites have been met.
1.2.3 The bridge maps this as now complete to Asset Transitions.
2 Asset Transitions moves to the next workflow step and coordinates power system startup.
2.1 Asset Transitions requests Power capability.
2.1.1 The bridge maps this to a demand on Power.
2.1.2 Power confirms it has a solution.
2.1.3 The bridge maps this as a confirmation to Asset Transitions.
2.2 Asset Transitions requests Propulsion capability.
2.2.1 The bridge maps this to a demand on Propulsion.
2.2.2 Propulsion confirms it has a solution.
2.2.3 The bridge maps this as a confirmation to Asset Transitions.
2.3 Asset Transitions determines that all solutions are in place.
2.4 Asset Transitions issues a demand that power is started up.
2.4.1 The bridge places a startup demand on Power.
2.5 Asset Transitions issues a demand that propulsion is started up.
2.5.1 The bridge places a startup demand on Propulsion.
Asset Transitions confirms power systems are go.
3 Asset Transitions moves to the next workflow step and issues startup demand to Inventory.
3.1 The bridge places a demand on Inventory.
3.1.1 Inventory draws power for Role Fit Discovery from Power.
Asset Transitions:Orchestrator
Interlocks:Atomic_Transaction_Pattern Bridge
Power_Systems:Business_Agreement_Pattern Bridge
Inventory_Check:Atomic_Transaction_Pattern Bridge
Interlocks:Partner
Power:Partner
Propulsion:Partner
Inventory:Partner
Initiate_interlock_check
determine_interlock_status : Specific_Instruction
Confirm_Go_State : Specific_Instruction
interlock_check_complete
power_requirement
determine_solution
determine_solution : confirmed
power_solution_confirmed
propulsion_requirement
determine_solution
determine_solution : confirmed
propulsion_solution_confirmed
power_startup
enact_solution
propulsion_requirement
enact_solution
initiate_inventor_check
specific_instruction

Figure 123: Orchestration Startup Example Sequence Diagram

A.3.1.5.3.1.4 Choreography

An important characteristic of the choreography interaction pattern is that it is about collaboration, not control. This interaction pattern is useful where the goal is to establish the rules and limits for interactions between collaborating components. Choreographies are often used between very disparate groups of components that operate autonomously and have not explicitly been designed to work together, for example an on-ground system of radars and facilities that may occasionally interact with air vehicles. They can operate at relatively specific levels of detail; for example informing components of which counterpart relationships to setup when a new store is attached to the air vehicle (e.g. different components are involved depending on whether it is a fuel tank or a missile).

The component that owns the choreography and the collaborating components need to be able to identify the context of the collaboration before the choreography transaction itself starts. Data loaded in the components from common or at least aligned Mission Plans enables the collaborators to register for involvement in the particular collaboration group, typically when a set of conditions are met (e.g. this may simply be at startup).

The transaction is initiated by the collaborators registering with the choreographer, and requesting the rules and limits. The collaborators then use the information supplied as the basis for direct interactions, as shown in Figure 124: Choreography Generic Example. The choreographer knows about the different groups involved, but is unlikely to control their actions.

As you might expect, the details of the rules can be quite complex, and there are several protocols designed explicitly to express this. See Ref. [41] for a comparison of three of the standards.

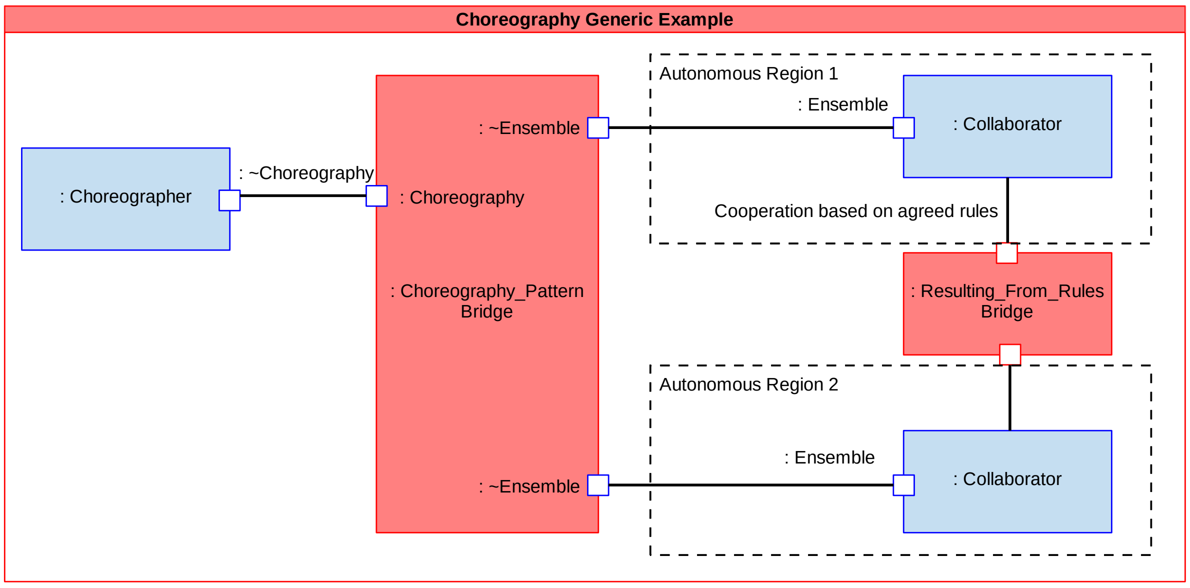


Figure 124: Choreography Generic Example

In the example shown in Figure 125: Choreography Refuelling Formation Example and Figure 126: Choreography Refuelling Formation Example Sequence, the air platform is required to perform an air-to-air refuelling (AAR). The Formations component can be used to set up a collaboration relationship between the Routes and Tactical Objects components, so as to enable Formations to coordinate a suitable rendezvous for refuelling. In this scenario, the tanker aircraft is not a PRA based platform and cannot accept, nor provide, manoeuvring control.

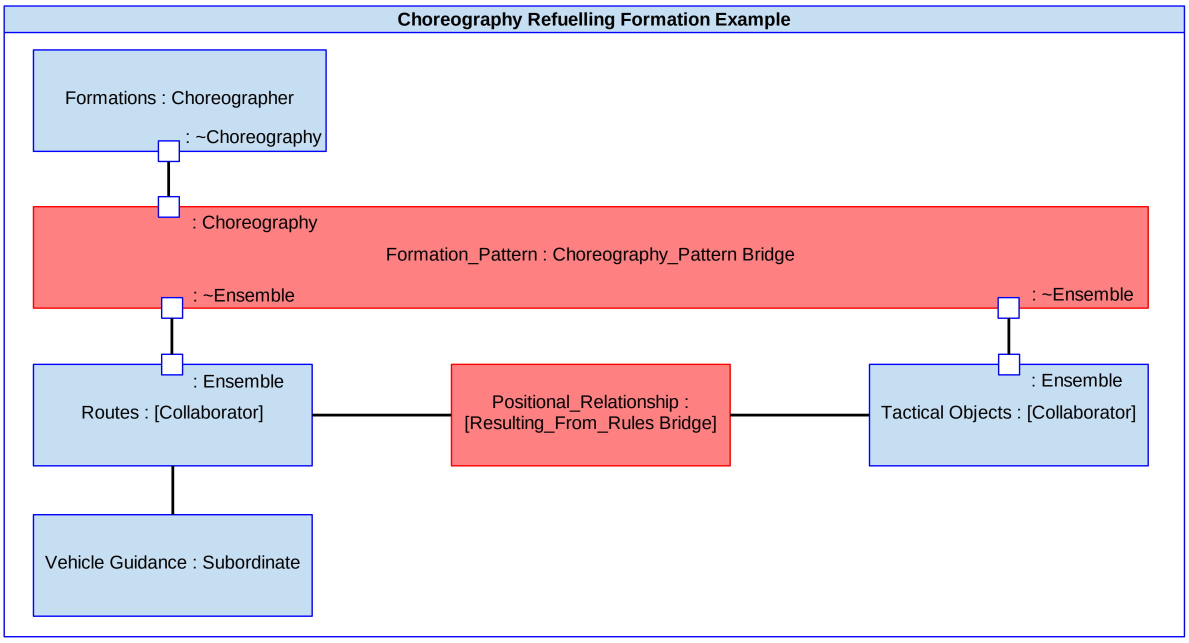

Choreography Refuelling Formation Example
Formations : Choreographer
: ~Choreography
: Choreography
Formation_Pattern : Choreography_Pattern Bridge
: ~Ensemble
: ~Ensemble
: Ensemble
Routes : [Collaborator]
Positional_Relationship : [Resulting_From_Rules Bridge]
: Ensemble
Tactical Objects : [Collaborator]
Vehicle Guidance : Subordinate

Figure 125: Choreography Refuelling Formation Example

The Formations component can provide the initial positional offset that the air vehicle should adopt before joining formation, and it is provided with an identifier for the tanker aircraft, but not its current position. Therefore two services are required, one that can route the air vehicle to a specified location, and another that can provide the position of the tanker aircraft to which a positional offset can be applied.

The Tactical Objects component has the capability to provide the tanker position, and the Routes component has the capability to direct the manoeuvring of the aircraft. These capabilities are reported to the Formations component via the bridge, which then allocates the service provider based on their stated capabilities.

Since the Formations component does not need to know the actual position of the tanker, the pattern can be used to instruct Tactical Objects to transfer the positional information directly to the Routes component. The Routes component can then use this information to generate a route to the rendezvous point and request the manoeuvres via commands to Vehicle Guidance.

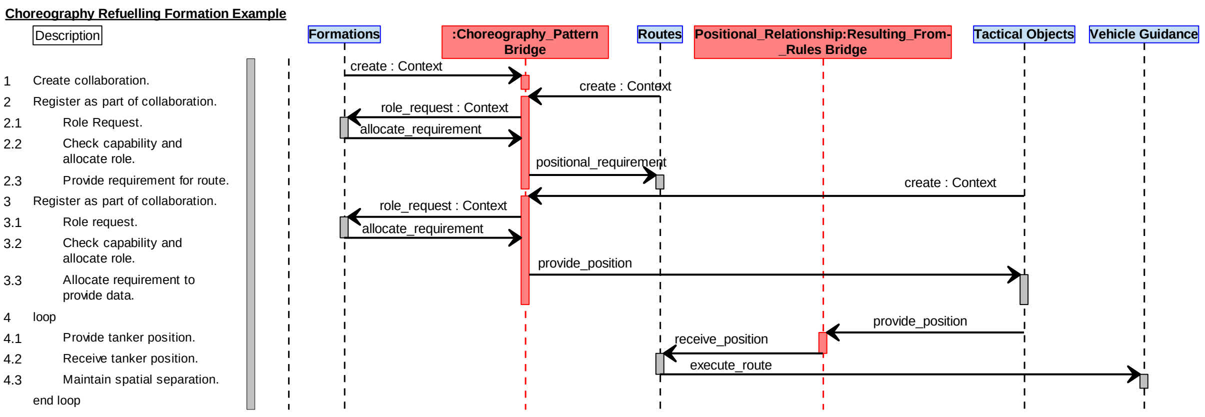

Choreography Refuelling Formation Example
Description
1 Create collaboration.
2 Register as part of collaboration.
2.1 Role Request.
2.2 Check capability and allocate role.
2.3 Provide requirement for route.
3 Register as part of collaboration.
3.1 Role request.
3.2 Check capability and allocate role.
3.3 Allocate requirement to provide data.
4 loop
4.1 Provide tanker position.
4.2 Receive tanker position.
4.3 Maintain spatial separation.
end loop
Formations
:Choreography_Pattern Bridge
Routes
Positional_Relationship:Resulting_From-_Rules Bridge
Tactical Objects
Vehicle Guidance
create : Context
create : Context
role_request : Context
allocate_requirement
positional_requirement
create : Context
role_request : Context
allocate_requirement
provide_position
provide_position
receive_position
execute_route

Figure 126: Choreography Refuelling Formation Example Sequence

Roles are allocated so that the required services are provided to Formations, and its consumption of services associated with the other components involved in the choreography is monitored. So long as the components have the capability to provide the required services, they can be left to manoeuvre the aircraft without further input from the Formations component.

Once the air vehicle reaches the location of the tanker aircraft, a new positioning requirement is generated and this relationship can be removed.

A.3.1.6 Component Connections Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Component Connections Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	Each PRA component represents a distinct subject matter area that is defined in its own language and scoped to have no knowledge of other components within the system. Consequently there are no shared interfaces between components, with bridges being used to close the semantic gap between components. Bridges can perform one or more of the following functions: • Data type conversion (i.e. translating between the different formats and measurements used by different components). • Semantic translation (i.e. translating the meaning of data in order to bridge the semantic gap between different component subject matters). • Invocation mapping between components (i.e. triggering behaviours in one or more components, as a result of event(s) generated in other components). Bridges should be kept as simple as possible, but not at the expense of polluting components, and should not contain any functionality that is defined as within the scope of a PRA component. The concept of counterparting is inherently embedded within the PRA by virtue of the subject matter separations that the PRA defines. Thus, a real world object or concept may be described across multiple PRA components with each component capturing a different aspect of the object or concept according to its unique subject matter. These different aspects have a natural relationship by virtue of the fact that they represent different aspects of a common real-world object or concept.	All PRA components are defined in their own language, consistent with their subject matter. PRA component definitions include component services which define what a component provides to the system, and consumes from the system, expressed in the components own terms. Bridges are used to connect components, enabling components to utilise one another's capability and information, whilst avoiding the pollution of subject matter between components. Bridges may also be used to connect components to other system elements, such as external equipment and middleware, where such interfaces would otherwise result in subject matter pollution, or when convenient to do so. The PRA defines a number of compliance rules relating to bridges. These rules ensure that bridges do not implement the functionality of PRA components and that component implementations do not incorporate the functions of bridges except where this is consistent with the subject matter of the component. For details of the compliance rules please see the PYRAMID Technical Standard, Ref. [1], section How to Comply with the PYRAMID Technical Standard. While the PRA defines a distinct set of component subject matters that necessarily imply counterpart relationships between different aspects of these subject matters, it does not explicitly define any such counterpart relationships.
Exploiter considerations	The following component connection related considerations are a matter for individual Exploiting Programmes: • The specification of bridges to achieve component connections (subject to the rules defined by the PRA). • The use of counterparts and methods for managing counterparts between components (within the bounds of the PRA subject matter definitions) including: • the explicit modelling of counterpart relationships. • the method used for the dynamic maintenance of counterparted objects. • Methods for defining interactions between components, such as the use of service contracts and component interaction patterns.	The PRA does not mandate a specific approach in respect of these considerations.

Table 21: Component Connections PYRAMID Concept Summary

A.3.2 Component Extensions

A.3.2.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

None

A.3.2.2 Introduction and Scope

This PYRAMID concept explains the concept of component extension, how it applies to the PRA and how the use of extensions supports the achievement of PYRAMID benefits. In the PRA, extensions can be developed to enhance the capabilities of PYRAMID components. This PYRAMID concept defines what is meant by an extension component, considers their benefits, provides guidance on their appropriate usage, and includes examples of extensions relating to the Tasks component.

A.3.2.2.1 What is Extension?

Extension is a design pattern and implementation mechanism used to open up a component into different parts so that behaviour of one part can be easily extended or specialised, reducing the impact of change. This technique enables behaviour to be factored into a separate component called an extension.

A.3.2.2.2 Why is Extension Important?

The ability to extend components is important as it is a key enabler for achieving PYRAMID benefits and as such, it is important to consider extensibility as part of component development. Extension components allow any given PYRAMID component to be extended in specific, well-defined ways that increase their adaptability, exploitability, future proofing and usability. This covers a variety of behaviour, including factoring out algorithms and catering for a range of special types. Factoring out behaviour in this way simplifies the component, allowing it to focus on the core behaviour without concern for special cases and allows changes to be restricted to extensions.

A.3.2.3 Overview

The PYRAMID concept for component extension can be summarised as follows:

- An extension component is a developed component that separates out or extends the functionality provided by a parent component whilst remaining within the subject matter of the PRA component the parent component is developed from.
- Any component can have extensions.
- A component is defined as an extension if only a parent component has access to its provided services.

- The PRA identifies some potential extension components (see Examples of Tasks Extensions); however, these are only examples and neither their use nor adherence to them is required.

A.3.2.4 Extension Definitions

The following definitions apply to extensions:

Parent Component: a developed component that supports the use of extension components. A parent component makes itself extensible by defining one or more extension points.

Extension Component: a developed component that separates out or extends the functionality provided by a parent component whilst remaining within the subject matter of the PRA component the parent component is developed from. An extension component's provided services fulfil the needs of an extension point.

Extension Point: a consumed service that defines the parent's dependency upon an extension for a single purpose.

Extension Set: a set of one or more extension components that satisfy the same extension point.

See Figure 127: Extension Definitions for a visual representation of these definitions.

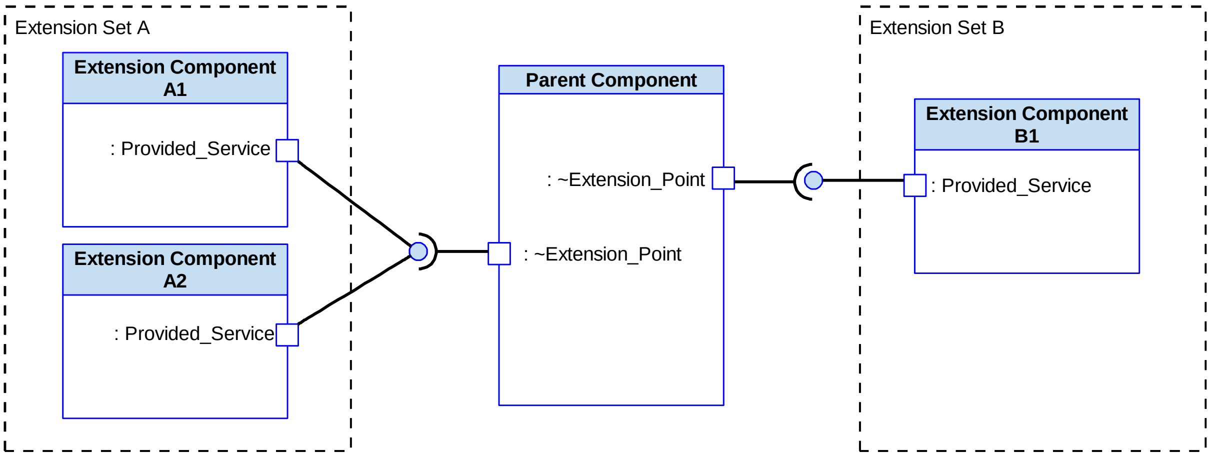


Figure 127: Extension Definitions

A.3.2.5 Criteria and Guidance for Extensions

An extension component is a PYRAMID component which adheres to the following rule:

- Only a parent component has access to the provided services of an extension component.

The rule does not influence PYRAMID compliance and an extension, like any PYRAMID component, is only required to adhere to the subject matter of the PRA component on which it is based. This is achieved through adherence to the responsibilities of the PRA component, as defined in the PYRAMID Technical Standard, Ref. [1] section PYRAMID Component Compliance Rules. Following the above rule for extension components merely determines if the component can be declared as being an extension component.

By imposing this restriction, the integration of different component variants, within the same deployment, that are based on the same PRA component is simplified. By ensuring that there is only one PYRAMID component, the parent component, that provides services to the wider system, the routing of component interactions can be simplified particularly when dealing with multiple extensions. Other issues, such as prioritising and scheduling (e.g. resource access), and handling conflicting demands, can be managed in a more deterministic way. This potentially comes at the expense of reduced performance, such as increased latency; although in many cases it may actually improve performance, by speeding up the resolution of conflicting demands on the set of PYRAMID components based on the same PRA component.

To ensure the benefits of parent-extension relationships, it is recommended that each instance of an extension component, within a system, provides services to only one parent component at any one time. Providing services to multiple parents at one time, may create conflicting demands on the extension component which would need to be managed, complicating the use of the extension. However, this recommendation does not restrict the following:

- Different instances of the same extension component supporting different parent components.
- The same extension component design/software being used to support different parent components in different systems or different interchangeable parent components for the same system.
- Different parent component instances or variants being present on the same system at the same time for redundancy purposes, where only one parent interacts with the extension component at any one time.

In addition to all provided services being consumed by the parent component, there will often be additional benefits to implementing an extension component's consumed services such that they are provided by the parent component. This can further reduce overall system complexity, as well as, in some circumstances, being necessary to support safety or security partitioning.

Figure 128: Extension Services shows how an extension component can consume services from another component, or from its parent. Whilst not shown, a parent component may also have other dedicated services with its extension components, such as to identify them and to configure its interface with them.

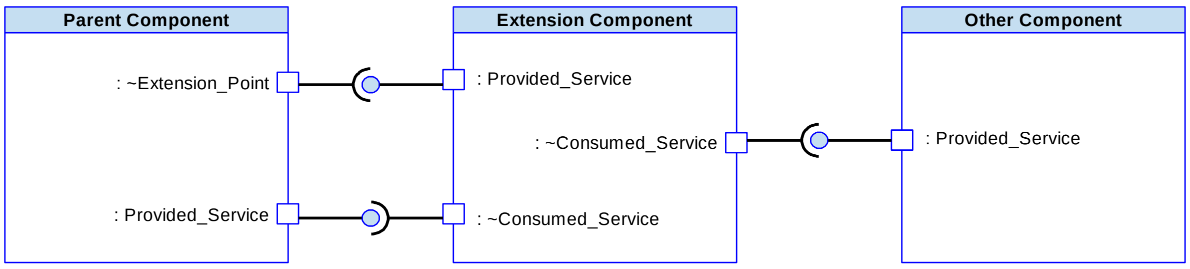


Figure 128: Extension Services

Unlike other PYRAMID components, it is not required that a bridge is used to connect an extension component to its parent component. This is because the interface remains within the subject matter of the same PRA component, allowing the interface of the extension component to be designed identically to its parent component. However, in many cases, it would be good practice to use a bridge, since this could reduce the need to modify an extension component if the parent component changes, and vice versa.

Any PRA component can be realised using extensions. Typical uses for extensions include supporting security, safety, or performance based partitioning, as well as having different extensions for different capability specialisations.

Extension components can be data-driven directly, like any PYRAMID component, rather than routing the data set through the parent component. Either approach is valid, since, for example, if multiple components require the same data set, it may be preferable to transfer the relevant data via the parent.

A.3.2.6 When Should Extensions be Used?

Exploiting Programmes should consider whether or not components would benefit from extensions when analysing component subject matters against their system requirements. Within each PRA components definition, there are design consideration suggestions for the potential uses of extensions for that component. It is highly likely that many components will benefit from using extensions. The Examples of Tasks Extensions section, shows different examples of extensions for a hypothetical PYRAMID Tasks component. However, these examples and suggestions do not constitute a PYRAMID requirement and thus extending these components is not a requirement for a deployment to be compliant with the PYRAMID Technical Standard.

The following might be reasons for using extensions:

- **Mission requirements:**

  Extensions can be used to support variations on mission requirements. Extensions can be used to capture the variations on the mission, e.g. performing a sensing mission or performing an attack mission. In this scenario, the functionality for mission specific elements is performed by the extension component and any core functionality is performed by the parent component. An example of this type is the Tasks component Attack extension, if the Tasks component is determining a solution that requires an attack to be performed the Attack extension can be used to determine that part of the solution.

- **A component has a super-type specialised by sub-types:**

  A component that defines a super-type can delegate the definition of subtypes to extensions, factoring out specialisation and opening up the component to cater for an extensible range of subtypes. This simplifies the parent, minimises the impact of change and makes its capabilities future proof. When new subtypes with special behaviour are discovered, they can be added through extension without change to the parent. For example, a component dealing with hardware would not want to define concrete classes for all devices currently required by the system. It would likely define a super-type of device that captures the common behaviour. The specific devices would be subtypes of the super-type and the subtypes would be defined in one or more extensions.

- **The component's functionality is:**

  - Complex - it may benefit a component to factor out complexity in order to isolate it. This will simplify the component and provide easier maintenance and testing for both the now simplified core logic and the isolated complex logic.

  - Optional - any behaviour that is optional can be factored out so that this behaviour is only present in the system when required. For example, if the platform is configured for reconnaissance missions and is not intended to perform attack missions, the Search extension is applicable to the platform but the Attack extension is unnecessary and can be excluded.
  - Open to alternative approaches - where alternative algorithms exist, use of extension can allow Exploiting Programmes to choose the algorithms that suit their needs. This enables a system to have a single algorithm where the choice is made at build time, or multiple options chosen at runtime by user selection or constraints. An example of multiple options would be multiple ways of calculating the same route, such as fastest, shortest or most fuel-efficient.
- **A component has a mix of requirements for:**
  - Safety integrity - it may be necessary to separate out behaviour based on safety integrity. For example, a parent component and some extensions may be required to be high integrity, but other extensions could be justified as lower integrity by the Exploiting Programme's safety analysis.
  - Data security - at times data can be highly sensitive and on a strict need to know basis. A component's core logic does not need to know the detailed data that is kept in an extension, meaning the definition and access to the data is kept discrete and knowledge of its detail can be removed from any system entirely by excluding the extension.

A.3.2.7 Examples of Tasks Extensions

The purpose of this section is to provide examples of component extensions, specifically extensions for a hypothetical PYRAMID Tasks component. These are examples of how extensions could be defined, rather than a comprehensive list of potential extensions. Exploiters will need to identify the specific application of extensions to meet the needs of their deployment.

The component extension definitions in this section have the following information:

- A Role, which is its purpose.
- An Overview, which defines the standard pattern of use, and examples of use. This describes, at a high-level, how the extension is used by the parent component to achieve goals of the parent component that relate to a specific specialised area of its subject matter.
- A set of Responsibilities, which describe the extended or specialised behaviour it provides to the parent component.
- A Design Rationale containing a set of assumptions, design considerations (including exploitation considerations), safety considerations and security considerations.

Note each tactic described below is just an example and their descriptions reference entities from the Tasks component subject matter semantics diagram. For further understanding, reference should be made to the Tasks component or extensions thereof.

Four example extension components are included to highlight the usage of extensions:

- Tactic: Attack
- Tactic: Communication
- Tactic: Search
- Tactic: Transit

These have been chosen on the basis that they each address a specific mission element that elicits different derived requirements and information needs, with the parent component selecting and overseeing the appropriate extension(s) for the phases of the mission. Note that these examples do not cover any demarcation of functionality between the parent and its extensions, or the selection of these extensions by the parent as appropriate. This detail would be subject to deployment specific design choices and exceeds the example extension component definitions.

Examples of usage of these extensions can be found in Appendix C: Interaction Views:

- Figure 200: Plan For A/S Engagement IV for an example usage of the Tactic: Attack extension.
- Figure 165: Network Initialisation IV for an example usage of the Tactic: Communication extension.
- Figure 172: Search IV for an example usage of the Tactic: Search extension.
- Figure 39: Setting up a Sensing Task PYRAMID concept diagram for an example usage of the Tactic: Search and Tactic: Transit extensions.

A.3.2.7.1 Attack

A.3.2.7.1.1 Role

The role of Attack is to determine and coordinate a set of delegated actions required to achieve an attack task.

A.3.2.7.1.2 Overview

Standard Pattern of Use

To achieve a solution the Tasks component delegates the relevant portions of the Taskings to the appropriate Tactic Extension. Attack manages the attack portion of Tasks Taskings and delegates a set of actions (e.g. provide destructive or suppressive effects in a military operation), including any supporting actions (e.g. communicate with ground forces to locate hostile entities) to be achieved by other components.

Examples of Use

When Tasks is defining a solution to a task that requires an attack to be undertaken, Attack can be invoked to undertake the determination and management of the attack portion of the provided Tasking. Scenarios where this may take effect can be seen where the:

- Exploiting Platform performs an attack where destructive or suppressive effects are needed in a military operation against hostile entity or entities.
- Use of a combat uncrewed aerial vehicle in an attack where the ownship detects a high threat hostile entity that must be eliminated.

A.3.2.7.1.3 Responsibilities

assess_attack_capability

- To assess the available attack task capability based upon the available composite capabilities and observed anomalies.

predict_attack_capability_progression

- To predict the progression of the ability to carry out the attack capability over time and with use.

determine_attack_implementation_solution

- To determine a Sequence of Derived_Needs for an attack using available attack Actions.

determine_attack_implementation_solution_costs

- To determine the costs of implementing an attack Sequence of Derived_Needs using the available capability.

satisfy_attack_dependencies_between_derived_needs

- To satisfy the attack dependencies between Derived_Needs by arranging their Sequence.

identify_missing_attack_information

- To identify missing information which could improve the certainty or specificity of the assessment of the ability to respond to a System_Stimulus.

coordinate_attack_task_enactment

- To coordinate the enactment of a task solution via the execution of Derived_Needs in a Sequence.

A.3.2.7.1.4 Design Rationale

A.3.2.7.1.4.1 Assumptions

- There will be significant variability in goals of an attack, their associated target numbers and types, the assets and capabilities available, and other operational details that need to be considered to prosecute an attack across different mission scenarios. Catering for all necessary combinations of these may require specific subject matter expertise; therefore it is assumed that producing an Attack tactic component extension, separate from a Tasks parent component, could be advantageous to a deployment.

A.3.2.7.1.4.2 Design Considerations

- Attack is an extension for the Tasks component.

A.3.2.7.1.4.3 Safety Considerations

The indicative IDAL is DAL B.

The rationale behind this is:

- Failure of this component could result in weapons impacting locations not intended by the crew and so result in unintended harm to third parties. This drives a DAL B indicative IDAL.

A.3.2.7.1.4.4 Security Considerations

This extension component will receive information and contains attack TTP appropriate for the mission being performed, therefore the extension can be assumed to have an indicative security classification of SNEO.

See the Tasks component for security functions.

A.3.2.7.2 Communication

A.3.2.7.2.1 Role

The role of Communication is to determine and coordinate a set of delegated actions required to achieve a communication task.

A.3.2.7.2.2 Overview

Standard Pattern of Use

To achieve a solution the Tasks component delegates the relevant portions of the Taskings to the appropriate Tactic Extension. Communication manages the communication portion of Tasks Taskings and delegates a set of actions (e.g. provide communication coverage in the required area), including any supporting actions (e.g. transit to the required area to provide the communication coverage) to be achieved by other components.

Examples of Use

When Tasks is defining a solution to a task that requires a communication to be undertaken, Communication can be invoked to undertake the determination and management of the communication portion of the provided Tasking, for example to:

- Provide communication coverage in the required area.
- Establish a communication plan to confirm when and where the ownship needs to change the communication capability during a mission.
- Ensure the requirement for two-way communication is active during critical points of a mission, if communication is lost corrective action should be applied.

A.3.2.7.2.3 Responsibilities

determine_communication_implementation_solution

- To determine a communication Sequence of Derived_Needs to fulfil an actual or potential communication need using communication and other Actions (e.g. movement, link and network).

determine_communication_implementation_solution_costs

- To determine the costs of communication against given Optimisation_Criterion.

predict_communication_capability_progression

- To predict the progression of communications capability over time.

assess_communication_capability

- To assess the available communications capability based upon available Action capabilities (e.g. network or links) and observed anomalies (e.g. changes in connection capabilities).

identify_missing_communication_information

- To identify missing information which could improve the certainty or specificity of the assessment of the ability to respond to a System_Stimulus.

coordinate_communication_task_enactment

- To coordinate the enactment of a task solution via the execution of Derived_Needs in a Sequence.

identify_communication_pre-conditions

- To identify the pre-conditions required between Derived_Needs in a Sequence (e.g. position and compatibility of communicating parties).

A.3.2.7.2.4 Design Rationale

A.3.2.7.2.4.1 Assumptions

- There will be significant variability in goals of a communication activity, the specific methods and protocols involved, the assets and capabilities available, and other operational details that need to be considered to establish and maintain communication across different mission scenarios. Catering for all necessary combinations of these may require specific subject matter expertise; therefore it is assumed that producing a Communications tactic component extension, separate from a Tasks parent component, could be advantageous to a deployment.

A.3.2.7.2.4.2 Design Considerations

- Communication is an extension for the Tasks component.
- Non-communications type actions might be needed to fulfil the communication task. For example, an inter-vehicle data link may not be possible to maintain if a mountain intersects the line of sight between air vehicles, so one or both air vehicles will need re-routing to ensure continuous communications.

A.3.2.7.2.4.3 Safety Considerations

The indicative IDAL is DAL C.

The rationale behind this is:

- Failure of this component may result in the inability to communicate between, for example, a ground based control station and the air vehicle. This is primarily a concern for UAVs, but may apply to manned air vehicles where some functions are controlled by external users. As loss of communications can occur frequently for reasons outside of the control of the air system (e.g. interference due to weather or satellite infrastructure) then the air vehicle will have been designed to mitigate a loss of communications. For a UAS this would be achieved by relying on pre-determined automatic / autonomous behaviour. For this failure mode it is concluded that failure of this component may result in a "significant reduction in safety margins", which has a major severity. Therefore, the indicative DAL is C.
- This component does not handle the data being transferred. Therefore, this component cannot corrupt data.

A.3.2.7.2.4.4 Security Considerations

This extension component will receive information and contains communications TTP appropriate for the mission being performed and methods of communications available, therefore the extension can be assumed to have an indicative security classification of SNEO.

See the Tasks component for security functions.

A.3.2.7.3 Search

A.3.2.7.3.1 Role

The role of Search is to determine and coordinate a set of delegated actions required to achieve a search task.

A.3.2.7.3.2 Overview

Standard Pattern of Use

To achieve a solution the Tasks component delegates the relevant portions of the Taskings to the appropriate Tactic Extension. Search manages the search portion of Tasks Taskings and delegates a set of actions (e.g. provide quality images of enemy threats), including any supporting actions (e.g. ensure to keep low observability) to be achieved by other components.

Examples of Use

When Tasks is defining a solution to a task that requires a search to be undertaken, Search can be invoked to undertake the determination and management of the search portion of the provided Tasking, it could be used to:

- Ensure the air vehicle follows a search pattern to collate imaging of any hostile enemies within a region of a search area.
- Perform a search in the area of interest for Combat Search and Rescue.
- Collect quality images of enemy hostiles that may be engaged during a subsequent mission.

A.3.2.7.3.3 Responsibilities

assess_search_capability

- To assess the available search capability based upon available Action capabilities and their observed anomalies.

determine_search_implementation_solution

- To determine a Sequence of Derived_Needs for a search task using available search Actions (including routing, sensing, and tactical data processing).

determine_search_implementation_solution_costs

- To determine the costs of a search against given Optimisation_Criterion.

predict_search_capability_progression

- To predict the progression of search capability over time.

identify_search_pre-conditions

- To identify the pre-conditions required between Derived_Needs in a Sequence.

identify_missing_search_information

- To identify missing information which could improve the certainty or specificity of the assessment of the ability to respond to a System_Stimulus.

coordinate_search_task_enactment

- To coordinate the enactment of a task solution via the execution of Derived_Needs in a Sequence.

A.3.2.7.3.4 Design Rationale

A.3.2.7.3.4.1 Assumptions

- There will be significant variability in goals of a search activity, the associated potential objects and the information to be gathered, the assets and capabilities available, and other operational details that need to be considered to carry out a search across different mission scenarios. Catering for all necessary combinations of these may require specific subject matter expertise; therefore it is assumed that producing a Search tactic component extension, separate from a Tasks parent component, could be advantageous to a deployment.

A.3.2.7.3.4.2 Design Considerations

- Search is an extension for the Tasks component.
- Search is required for search type tasks that combine sensing activities with other Actions to detect, perform geolocation, recognise, identify, or otherwise increase knowledge of objects or features in the tactical environment. For example, locate all the tanks in a region.

A.3.2.7.3.4.3 Safety Considerations

The indicative IDAL is DAL C.

The rationale behind this is:

- Failure of this component would result in the inability to search, which does not impact safety. It is expected that if this component attempted to initiate search activities which compromised safety (e.g. intended route impacted terrain) that other, high integrity, components (e.g. Interlocks) would prevent any unsafe Actions from being performed. Therefore, an IDAL no more onerous than DAL C, is considered appropriate for this component.

A.3.2.7.3.4.4 Security Considerations

This extension component will receive information and contains search TTP appropriate for the mission being performed, therefore the extension can be assumed to have an indicative security classification of SNEO.

See the Tasks component for security functions.

A.3.2.7.4 Transit

A.3.2.7.4.1 Role

The role of Transit is to determine and coordinate a set of delegated actions required to achieve transit tasks.

A.3.2.7.4.2 Overview

Standard Pattern of Use

To achieve a solution the Tasks component delegates the relevant portions of the Taskings to the appropriate Tactic Extensions. Transit manages the transit portion of Tasks Taskings and delegates a set of actions (e.g. transit air vehicle between areas to designation), including any supporting actions (e.g. ensure route is viable) to be achieved by other components.

Examples of Use

When Tasks is defining a solution to a task that requires a transit to be undertaken, Transit can be invoked to undertake the determination and management of the transit portion of the provided Tasking, It could be used to:

- Transit the air vehicle between areas during mission to reach the located designation.
- When called upon for close air support the air vehicle will transit between areas (e.g. from the flight hub to the battlefield).

A.3.2.7.4.3 Responsibilities

determine_transit_implementation_solution

- To determine a Sequence of Derived_Needs for a transit task using available transit Actions to reposition the air vehicle.

determine_transit_implementation_solution_costs

- To determine the costs of transit against given Optimisation_Criterion (e.g. fuel use, safety margin or level of authorisation required).

assess_transit_capability

- To assess the available transit capability based upon available Action capabilities and observed anomalies.

predict_transit_capability_progression

- To predict the progression of transit capability over time.

coordinate_transit_task_enactment

- To coordinate the enactment of a task solution via the execution of Derived_Needs in a Sequence.

identify_missing_transit_information

- To identify missing information which could improve the certainty or specificity of the assessment of the ability to respond to a System_Stimulus.

A.3.2.7.4.4 Design Rationale

A.3.2.7.4.4.1 Assumptions

- There may be significant variability in a transit activity, the airspace rules to be complied with, the navigation assets and capabilities available, and other operational details that need to be considered to carry out transit activities across different mission scenarios. Catering for all necessary combinations of these may require specific subject matter expertise; therefore it is assumed that producing a Transit tactic component extension, separate from a Tasks parent component, could be advantageous to a deployment.

A.3.2.7.4.4.2 Design Considerations

- Transit is an extension for the Tasks component.
- Transit is required for tasks that reposition the air vehicle so that it can perform another task.

A.3.2.7.4.4.3 Safety Considerations

The indicative IDAL is DAL B.

The rationale behind this is:

- Failure of this component could result in an inappropriate transit action being carried out. As it is expected that other components (e.g. Routes) would ensure the transit route is achievable (in terms of fuel, etc.), this is considered a "large reduction in safety margins" (critical severity) and so the indicative IDAL is DAL B.

A.3.2.7.4.4.4 Security Considerations

This extension component will receive information and contains transit TTP appropriate for the mission being performed, therefore the extension can be assumed to have an indicative security classification of SNEO.

See the Tasks component for security functions.

A.3.2.8 Component Extensions Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Component Extensions Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PRA supports the use of component extensions and any PYRAMID component may utilise extensions. An extension implements a subset of the PRA components defined subject matter. By definition, an extension's provided services are only accessible from the parent component. Extensions may include consumed services that access the services of other components directly. The provided services on a parent component, and the consumed services on the parent or extension, connect to other components using bridges. Services connecting a parent and extension are internal to the PRA component definition and such connections may or may not utilise bridges. Extensions may include services that interact across the PRA scope boundary, where these are required to fulfil a responsibility defined for the associated PRA component. Extensions can be data driven like any PYRAMID component.	In principle any PRA component may be implemented using extensions, however the PRA neither mandates the use of extensions nor defines any internal component structures. The PRA therefore does not define any specific extensions, but does identify where components might benefit from the implementation of extensions (see the 'Design Considerations' section of each component definition). Examples of the application of extensions for the Tasks component are provided in section Examples of Tasks Extensions.
Exploiter considerations	The following component extension related considerations are a matter for individual Exploiting Programmes: • The application of component extensions to any particular PYRAMID component development. • Whether bridges are used in connecting the parent to the extension.	The PRA does not mandate a specific approach in respect of these considerations.

Table 22: Component Extensions PYRAMID Concept Summary

A.3.3 Data Driving

A.3.3.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

None

A.3.3.2 Introduction and Scope

This PYRAMID concept explains how data in the form of data sets can be applied to components to modify the component behaviour, including the behaviour at its interfaces, to cater for different system configurations, different specialisations of capabilities, and performance optimisation.

A.3.3.2.1 What is Data Driving?

Data driving is the act of applying a data set to a component design, in order to provide a complete definition of how the component is required to behave. This requires the component developers to establish parameters, or settings, within a component design that have values defined by the data set. These data sets are sometimes referred to as data files, configuration files, configuration data files, or configuration data.

A.3.3.2.2 Why is Data Driving Important?

Data driving enables component or system behaviour to be tailored, without having to redevelop component designs, to:

- Cater for different system configurations, including newly required configurations (e.g. role fit configurations).
- Specialise capabilities for specific situations, including newly required situations (e.g. specialising sensor capabilities for a specific operating environment).
- Optimise capabilities based on emerging information, such as trial feedback (e.g. general sensor performance optimisation).

This provides the opportunity for new user operational requirements and performance improvements to be achieved more rapidly, and to be achieved by only involving the essential stakeholders.

A.3.3.3 Overview

The process of applying data driving can be summarised as:

- Any component required to utilise data driving must be designed with its relevant parameters or settings being data drivable.
- A separate data set (which is sometimes referred to as a data file, a configuration file, a configuration data file, or configuration data) specifies the values for these parameters.

- The data set can be applied to the component either before compiling the component executable software, causing the data to become part of the executable software, or the data set can be applied after compilation.

This allows independent development of the component design from the data set design, which enables the component behaviour that is influenced by the data sets design to be refined or changed without changing the component design.

A.3.3.4 Types of Data Driving

The data used for data driving can be categorised as either:

- Build time data
- Runtime data

The distinction between these types is whether or not the data becomes incorporated within the executable software. The distinction has implications on when the data needs to be created and utilised. However, it should be noted that either type of data may be created and used for testing purposes at any point prior to when the data is required to be applied, such as to support testing of individual components or a system design within a model.

Although both types of data driving can generally be used for similar purposes this is not always the case and some uses of data driving are more suited towards a certain type of data driving.

A.3.3.4.1 Build Time Data

Build time data describes data that forms part of the compiled executable software component. It must, therefore, be developed and applied to the component design prior to compiling the executable software.

Therefore, applying different build time data sets to a component is one method of creating different component variants.

The build time data type of data driving is most likely to be used where greater control over the software capabilities is required by the software developer and where a lower rate of change is needed, such as when defining some types of vehicle specific data (e.g. coefficients within vehicle control laws) or details influenced by the execution platform (e.g. timeouts).

A.3.3.4.2 Runtime Data

Runtime data is used by the compiled executable software, and, therefore, does not form part of the executable software. Except for early testing purposes of a modelled system design, runtime data is applied after software compilation to the executable software.

The runtime data type of data driving is most likely to be used where less rigorous control over the software capabilities is required by the software developer and where a higher rate of change is needed, such as when defining data that changes based on the particular vehicle configuration, mission type, operating region, or adversary, or where data will require frequent refinement based on the results of flight trials. For instance, this could include data that optimises sensor performance (either for a specific mission requirement or based on trials feedback).

Runtime data may include various aspects of mission data, either single mission data or multi mission data; while other aspects of mission data, such as a preloaded route, may not be classified as data driving. There is no distinct rule that unambiguously differentiates what should and should not be classed as data driving; however, the distinction is typically unimportant and there is little harm in treating all forms of mission data as if they are runtime data.

A.3.3.5 Uses of Data Driving

There are a large number of ways that data driving can be used. However, how data driving can be used can be broadly categorised as:

- Modifying the core component behaviour or customising its capability.
- Modifying how the component interacts at its interfaces.

A.3.3.5.1 Modification to Behaviour or Customisation of Capability

Modifying behaviours or customisation of capabilities can allow a component or system to be adapted to suit specific engineering or operational needs.

By modifying behaviour and customising capability, data driving can enable:

- Customisation of a component to cater for specific execution platform details or specific system details. This allows:
    - The same core component to be used for different execution platforms or systems. The data-driven elements may not necessarily define all of the platform specific variables, but they may help to define key parameters (e.g. specifying timeouts). This type of use is typically only applicable to build time data.
- Capability to be optimised or specialised. This allows capabilities to be tailored for:
    - Specific operating requirements (e.g. specific missions, types of mission, and operating environments). For example, the sensor capability may be optimised for detecting specific object types within a specific environment.
    - Specific vehicle or equipment configurations (e.g. an ISR, air to air, or air to surface configuration utilising different role fit sensors and countermeasures equipment).
- Capability to be enabled or disabled. This allows:
    - The same component to be used by different users, such as different nations, with different requirements.
    - Capabilities to only be active during particular scenarios (e.g. status reporting to telemetry systems only being active during flight trails and evaluation).
    - Capabilities only applicable to specific vehicle or equipment configurations only being active when they are needed.

A.3.3.5.2 Modification to Implemented Interfaces

Modifying the details of an implemented interface for a component through data driving can allow a component to be more easily integrated into a variety of systems, without needing different component designs. This does, however, require suitable provisions to be incorporated in the component design to be able to meet the different requirements of the different systems.

By modifying the details of interfaces, data driving can enable:

- Access to a capability through the interface to be enabled or disabled, providing an alternative way to enable or disable a capability. (Enabling and disabling capabilities is described within Modification to Behaviour or Customisation of Capability).
- Alternative forms of interaction, based on different interface paradigms, to be selected for use (e.g. selecting interactions that are service based or non-service based, periodic or aperiodic, etc.).
- Alternative representations of information at the interface to be defined or selected, such as specific data types or representations of information in alternative coordinate systems (e.g. using Cartesian instead of polar). This can allow closer alignment to other parts of the system (e.g. by adhering to the use of a standard data model).

This use of data driving may be of particular benefit where the same component is used on different systems with different interface requirements, especially when hosted on equipment that is required to be interoperable with different systems. For example, role fit equipment (such as a podded sensor or weapon) may be required to be used on different air vehicles, requiring different interfaces, without any modification to the equipment or its software.

Since components should remain independent of each other, this use of data driving does not remove the need for bridges; however, when used in appropriate circumstances, it may reduce the complexity of bridges and could potentially reduce the likelihood of integration problems.

A.3.3.6 Data Driving Considerations

There are a number of considerations that should be taken into account when determining what to apply data driving to and how to best apply it. This section highlights many of these considerations. Except where stated otherwise, the considerations are applicable to both types of data driving (build time data and runtime data), although their significance to one type or the other will vary.

A.3.3.6.1 Data Sets

Different data sets may be developed throughout the system lifecycle (potentially varying in maturity and developer) to adapt to evolving system requirements and operational needs.

Different data sets may be needed to:

- Allow a data set to be matured, including as a result of testing of the data set or obtaining updated source information that the data set is developed from.
- Support different stages of the system development and in service lifecycle. For build time data, different versions may be used to support component modelling or component compilation; whereas for runtime data, different versions may be used to support component modelling, component testing, system ground testing, system flight testing, or in service operation.

- Optimise or configure the system in different ways, providing a choice of data sets that are available for use based upon different mission needs.
- Introduce new capabilities, such as the integration of a new weapon type onto an air vehicle (where the air vehicle is designed to cater for different weapon types, within a general weapon class, through data driving).

The different data sets will often require appropriate configuration management to be in place. This is needed to ensure that they are used for the correct purpose and only used in conjunction with the appropriate versions, variants, or instances of components.

A.3.3.6.2 Data Set Developers

A data set does not have to be developed by the same developer as the software, and different data sets may have different developers.

Build time data is typically developed by industry, whereas runtime data can be developed by industry or by the end user.

The separation of the data set development from the component development potentially opens opportunities for allowing multiple stakeholders, with different expertise, to contribute to the overall capability of a component, without necessarily having to disclose proprietary or sensitive information.

A.3.3.6.3 Testing, Qualification, Verification and Certification

A data driving data set may need to be tested, qualified, verified, certified, or validated; therefore it should be considered when and how this should be done. (For brevity qualification, verification, certification, and validation are collectively referred to as ‘certification’ within this section; however, their distinction is still important to consider.) This will depend on a number of factors, including what the data is being used for and any associated safety, security, reliability, or performance requirements.

By their nature, build time data and the component executable software are typically tested and certified together, although there are opportunities to perform initial testing and assessment activities of these independently, such as model testing prior to the software being compiled.

Components or PYRAMID deployments may be certified independently from runtime data sets, or runtime data sets may form part of the component or system certification. This influences when and how the different data sets, including new data set versions and variants, are tested and certified. Likewise, it can influence the need to retest and recertify components or systems when new data sets are made available.

A.3.3.6.4 Default Settings and Overriding Default Settings

Runtime data may be used to override the default settings within a component, meaning that the data set does not necessarily need to be loaded, or even have been developed, in order for the component executable software to run correctly. In this case, the runtime data allows basic or generic capability enhancement or specialisation when needed.

Build time data may be used to define the default settings used within the component executable software. It is therefore possible that data sets for build time data and runtime data could data drive the same settings.

A.3.3.6.5 Data Set Confidentiality and Integrity

A data set may need to be protected against:

- Unauthorised parties being able to discern the information contained within the data set.
- Undetected modification or corruption of the data. The protection may also support error correction.

The methods used to achieve these may be required to safeguard classified or proprietary information, to prevent unauthorised or accidental changes, or prevent data corruption going undetected. Security measures, such as encryption and access controls, should be implemented where necessary to maintain data integrity and confidentiality throughout the development and operational lifecycles of the system and data set.

Whilst limiting access to data sets offers intellectual property and security protections, it may limit customisation options or reduce the speed at which changes to the data set can be made.

A.3.3.6.6 Safety, Security and Performance Deployment Considerations

The data used for data driving may contribute to system safety, security, or performance. Likewise, the standards, processes and guidelines being used for an Exploiting Programme may limit what can and cannot be done in terms of data driving.

These factors will influence the choice and implementation of data driving strategies.

This includes considerations about:

- Whether data driving is appropriate or allowed.
- The best type of data driving.
- The appropriate uses of data driving.
- The approach to testing, qualification, verification, certification, and validation.

A.3.3.6.7 Deployment Approaches to Implementing Data Driving

There are a variety of approaches to implementing data driving within components, and the most suitable approach will be influenced by the overall software development methodology being used. The PRA does not require any specific approach to be adopted.

An example of how data driving could be implemented in components, when using an object orientated approach, is to incorporate distinguishable specification classes into the design. The different specification classes contain the parameters that are data-driven, allowing the location and use of the data-driven data within the design to be easily understood and managed. For example, different instantiations of the specification classes (resulting from different data-driven data sets) could relate to different missile types that can be catered for within the design in a similar way. This would enable the relevant performance parameters (e.g. maximum intercept ranges) and applicable capability settings of the missile types to be catered for. Further information about the concept and use of specification classes can be found in Ref.[6].

Any PRA components for which data driving is an effective strategy for mitigating anticipated changes will have this rationale captured within the Assumptions and Design Considerations sections of their component definition. Note that no services are defined for data driving within the PRA (for further details see PYRAMID Technical Standard, Ref. [1], section Component Services Not Defined by the PRA).

A.3.3.7 Data Driving Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Data Driving Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PRA has been designed to support data driving and this is achieved by component designs, based on the PRA, making the relevant parameters, or settings, data drivable as required to meet the needs of an Exploiting Programme. Data driving may be applied to components before compiling the component executable software, causing the data to become part of the executable software and enabling the creation of different component versions or variants for specific deployments/exploitations. Data driving may also be applied to the component after compilation enabling customisation of components for specific operations/missions.	In principle any PRA component supports data driving, implemented as build time data or runtime data. While the appropriate use of data driving is recognised as good practise, the PRA does not mandate the use of data driving or any particular approach to data driving. Furthermore, the PRA does not define any component responsibilities or services in relation to data driving. The PRA identifies components where data driving is expected to be an effective strategy and in each case how data driving might be applicable (see the Design Considerations section of the component definitions).
Exploiter considerations	The following data driving related considerations are a matter for individual Exploiting Programmes: • How or when to apply data driving, e.g. as build time data or runtime data. • The choice of component interfaces and behaviours that should be modifiable through data driving. • How and when data driving data sets should be tested, qualified, verified, certified or validated. • If and how data driving data sets should be disclosed and protected. • Data driving strategy in relation to product safety, security and performance. • The provision of services to support data driving.	The PRA does not mandate a specific approach in respect of these considerations.

Table 23: Data Driving PYRAMID Concept Summary

A.4 Safety and Security PYRAMID Concepts

This section contains the Safety Analysis and Security Approach PYRAMID concepts. These explain what safety analysis and security approach means in the context of the development of the PRA components, describing the safety process that has been followed in assigning indicative design integrity to components, and providing support to Exploiting Programmes with the security accreditation process.

A.4.1 Safety Analysis

A.4.1.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None

Read in conjunction with

None

A.4.1.2 Introduction and Scope

This PYRAMID concept explains what safety analysis means in the context of the development of the PRA. For the PRA there is no safety risk. This is because the Exploiting Programme will be entirely responsible for demonstrating the Exploiting Platform meets the safety targets applicable to the Exploiting Platform. It describes the process that has been followed in assigning an indicative design integrity to components and the context (i.e. the assumptions) under which that process has been carried out. The design integrity of a component determines the level of rigour required during software development in order to reduce the safety risk to an acceptable level. The indicative design integrities are not requirements on an Exploiting Programme.

A.4.1.2.1 What is Safety Analysis?

An artefact is safe if it poses an acceptable risk of harm (to people) and damage (to property). Safety analysis is the examination of an artefact to determine the level of (safety) risk it poses.

Safety analysis is dependent on a definition of the Exploiting Platform and context it operates within. It is not feasible to have a PRA safety analysis that covers all possible exploiting cases. Therefore, the PRA safety analysis has assumed a conservative context, but aims to avoid being driven by cases that are considered extremely unlikely (e.g. non-typical Exploiting Platform types or extreme operating scenarios). This is to improve the quality of the PRA. Therefore:

- The Exploiting Programme will be entirely responsible for demonstrating the Exploiting Platform meets the safety targets applicable to the Exploiting Platform.
- The safety considerations contained in the PRA are not requirements on an Exploiting Programme nor are they expected to contribute to the Exploiting Programme's safety case.

A.4.1.2.2 Why is Safety Analysis Important?

Safety analysis is important because new and modified air systems are required to be safe in order to be certified, meet legal obligations and meet moral expectations.

A.4.1.3 Overview

The PYRAMID concept for safety analysis can be summarised as follows:

- All components have been assigned an indicative design integrity based on an analysis of (safety) risk within a defined context. These are not requirements on an Exploiting Programme.

A.4.1.4 Integrity Classifications

The indicative design integrities assigned to each component have been classified as Item Development Assurance Levels (IDALs) based on the civil aerospace standards ARP4754A Ref. [27] and ARP4761 Ref. [26]:

- DAL A
- DAL B
- DAL C

Notes:

- An indicative IDAL of C, covers C, D or E as the analysis has not attempted to differentiate between the lower DAL levels as it was judged that it would not drive the PRA.
- IDALs are determined during the safety analysis process (e.g. using ARP4754A Ref. [27] and ARP4761 Ref. [26]). The level of rigour required during software development is defined in DO-178C Ref. [17]. DO-178C uses the term 'software level', which is synonymous with IDALs. For software development, IDALs A, B and C correspond to Software Levels A, B and C.

A.4.1.5 Risk Acceptance Criteria

The risk acceptance criteria and severity classifications used are:

- **Catastrophic**: Failure conditions that result in the death of one or more people. Catastrophic outcomes are considered DAL A within the PRA safety considerations.
- **Critical**: Failure conditions that result in major injury to people, loss of aircraft or a large reduction in safety margins; critical outcomes are considered DAL B within the PRA safety considerations.
- **Major**: Failure conditions that result in minor injury to people, major damage to the aircraft or a significant reduction in safety margins; major outcomes are considered DAL C within the PRA safety considerations.

Notes:

- Where the failure of components can cause weapons to impact targets not intended by the crew, resulting in unintended harm to third parties, it is assumed this would drive an IDAL of

B. The consideration of other accidents, such as impacting the host air vehicle during release, may drive a more onerous DAL.

- These risk acceptance criteria are assumptions on what would be appropriate for a UK military air system. An Exploiting Programme would agree the risk acceptance criteria for their project with the purchaser.

A.4.1.6 Safety Analysis Context

The context assumed for the safety analysis is:

- The air vehicle is large (>5700kg) (based on the Defence Standard 00-970 Part 9 Ref. [11] definition).
- The air vehicle may be crewed.
- The air vehicle may be uncrewed.
- The air vehicle may overfly populated areas.

The assessment of individual components includes additional component specific context in the rationale where necessary.

A.4.1.7 Determination of the Indicative IDAL

The components have been assigned an "indicative IDAL" of A, B or C based on the risk acceptance criteria and context defined above. The indicative IDAL and supporting rationale for each component is documented in the "Safety Considerations" section of each PRA component definition.

- The indicative IDALs reflect the likely criticality of a particular component and so take credit for external independent events or failures of other components.
- IDAL assignment has been largely based on engineering judgement and experience. The rationales are as concise as possible.
- IDAL assignment has been informed by the systematic safety analysis previously performed during the development of the PRA. However, the safety consideration rationales are self-contained and so do not reference any separate material that was produced to support the development of the PRA.
- IDAL assignment has focused on the most severe safety impact of the component (i.e. captures the most severe Functional DAL for the component). Other, non-driving, cases which may apply to some air vehicles or operational roles are generally not discussed.

A.4.1.8 Safety Analysis Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Safety Analysis Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PYRAMID concept does not identify any generic component behaviours in relation to safety.	The PRA identifies relevant safety considerations within the 'Safety Considerations' section of each component. The PRA also assigns an indicative IDAL classification to each component based on an analysis of (safety) risk within a defined context. These indicative classifications are not requirements on an Exploiting Programme.
Specific PRA Component support for the PYRAMID Concept	The PYRAMID concept does not identify any specialised subject matters in relation to safety.	N/A.
Exploiter considerations	The following safety analysis related considerations are a matter for individual Exploiting Programmes: • Establishing applicable safety standards. • Defining the Exploiting Platform operating context for the purpose of safety analysis. • Agreement of risk acceptance criteria and determination of component IDALs. • The specific approach to system partitioning to meet safety requirements, for example, the use of component variants and instances, and the use of extensions; e.g. to isolate high IDAL functions or provide redundancy. • Demonstrating that safety targets have been met. • Demonstrating that the Exploiting Platform meets the safety targets applicable to the Exploiting Platform.	The PRA does not mandate a specific approach in respect of these considerations.

Table 24: Safety Analysis PYRAMID Concept Summary

A.4.2 Security Approach

A.4.2.1 Pre-Reading and Related PYRAMID Concepts

Prerequisite

None.

Read in conjunction with

- Recording and Logging
- Cyber Defence
- Safety Analysis

A.4.2.2 Introduction and Scope

This PYRAMID concept explains what is meant by the PRA security approach.

PYRAMID is intended to be incorporated on a range of platforms, with different infrastructures and security environments, therefore the security approach is designed to support Exploiting Programmes with the accreditation process whilst remaining flexible and with no specific security architecture being mandated.

As the PRA is a framework that specifies components that will exist at the application layer, it cannot specify an accreditable solution without the support of security functionality offered by the execution platform or externally to the system. Since they are defined independently of any Exploiting Platform or execution platform, without any understanding of the security environment, the PRA component definitions have no inherent security risk. Thus, the Exploiting Programme will be responsible for specifying the security environment and the security risks to be considered, and for demonstrating that the Exploiting Platform meets the security targets applicable to its operational use.

This PYRAMID concept explains the approach taken in order to identify the security considerations for each PRA component that support Exploiting Programmes with the accreditation process.

A.4.2.2.1 What is the Security Approach?

The PRA approach to security is that an assumed software and hardware infrastructure will provide many of the necessary security controls, atop of which will reside the components within the application layer. These components may contain additional security functions to support a multi-level integrated cyber defence.

Placing security functions within the components without the support of trust and assurance from the execution platform would be difficult to accredit; thus the PRA expects lower-level security to be present, including security partitioning, VPNs, gateways and trust anchors back to the hardware (an established authoritative entity for which trust is assumed, not derived). However, these lower-level security functions may also require security-related metadata provided by PYRAMID components in order to operate.

The approach taken for the PRA component analysis does not provide detailed information on how these or any other security requirement may be satisfied, nor does it provide any information on how any non-technical security controls are addressed. These will be covered by the full security analysis to be undertaken for the Exploiting Platform.

The security approach taken for the PRA has been to look at the components and how they might support the accreditation goal, with each component including Security Considerations that:

- Identify security related and enforcing functions that may be required or provided by the components, see the Security Functions section for further details.
- Identify an indicative security classification for the components and therefore the security domain that the component may reside within, see the Security Domains section for further details.

As the Exploiting Programme will have its own security concerns and security architecture the indicative classification is meant only as a guide to support security domain segregation, and is not a requirement on the Exploiting Programme.

A.4.2.2.2 Why is the Security Approach Important?

To achieve security accreditation of any system, the integrator has to generate a security accreditation case based on a formally documented understanding of the security aspects of the system. As the security environment for each PYRAMID deployment will be different, so each accreditation case will be different, but should cover topics including the operational context and the potential security risks the system could face. The risk assessment method can be as described in HMG IA Standard No. 1 & 2, Ref. [36], ISO/IEC 27005, Ref. [37], or the NIST Framework for Improving Critical Infrastructure Cybersecurity, Ref. [38], or as agreed with the security assurance coordinator for the Exploiting Programme.

Security risk mitigation will be through a series of security controls. Generic sets of security controls can be found in ISO27002, Ref. [39], ISO/IEC 15408, Ref. [40], and the NIST Framework for Improving Critical Infrastructure Cybersecurity, Ref. [38]; these will need to be tailored and are typically applied using a "defence in depth" approach, covering an array of personnel, physical, procedural and technical security controls to identify, protect, detect, respond and recover from a cyber attack. In practice, the controls identified will go beyond the scope of what can be achieved within PYRAMID components, which can only address a subset of the technical controls. A PYRAMID deployment is therefore dependent on further external controls to provide this defence in depth.

For accreditation the Exploiting Programme will need to demonstrate:

- That there has been a formally documented approach to identifying security vulnerabilities.
- Any identified vulnerabilities have been mitigated as far as reasonably possible.
- Any classified information is suitably partitioned and protected from attacks on its confidentiality, integrity and availability (see the Security Attributes section).
- Safety and airworthiness elements of the system are protected from malicious attack.
- It is able to generate (and rapidly regenerate when required due to system reconfiguration/update) the information or metadata required to configure the system’s security functions.

- It is able to generate (and rapidly regenerate when required due to system reconfiguration/update) the evidence required for certification.

The Security Considerations provided for the PRA components provide an indication of how they can support this process of accreditation through the development and documenting of security functions by:

- Identifying security functionality within components to act as risk mitigation controls.
- Supporting security partitioning by indicating the indicative classification of components.
- Identifying safety related functionality that should be protected.

Additionally, an indication of data flow is provided through the component service definitions.

Neither the PRA nor any standalone component can be considered accreditable in its own right, as this requires the context of the supporting infrastructure and security environment of the Exploiting Programme.

A.4.2.3 Overview

The PYRAMID concept for describing the security approach can be summarised as follows:

- PRA components are assigned an indicative security classification that corresponds to the appropriate levels of control that should be put in place to identify, protect, detect, respond to and recover from security threats. The assigned classification is based on an assessment of generic security risk within the context of a military security environment. These are not requirements on an Exploiting Programme.

The key points to be taken from the PYRAMID concept include:

- The security approach is flexible enough to be incorporated in a number of security environments.
- PYRAMID components cannot provide a complete security solution for the Exploiting Programme.
- Component developers and the integrators of a PYRAMID deployment will need to include configuration and metadata in their components in order that the underlying infrastructure can implement effective security controls and establish chains of trust, etc.
- The security approach is more than just protecting classified information but should also consider protecting the safety, operational advantage, freedom of operation and reputation of the entities that operate the Exploiting Platform and the intellectual property of its developers.
- The components contain a number of security functions (either SEFs or SRFs) from which a level of development rigour might be assumed, however the level of development rigour is ultimately dependant on the security environment and security requirements of the Exploiting Programme.

A.4.2.4 Determination

The determination of the component's security functions and indicative classifications has been based upon the assessment of a typical military platform, as well as assessment of the types of interactions defined in the IVs, the likely algorithms involved and the data that may reasonably be expected to be exchanged.

Any specific assumptions relevant to this assessment performed are included in the general Assumptions section of the PRA component definition's Design Rationale section. For example, this may include any instances whereby one component relies on another component for performing a required security-related activity, such as encryption, or other applicable component behaviour and/or knowledge, etc.

The indicative classification has been stated at the lowest level that might be reasonably expected, often with a caveat that, in certain situations, this can be higher. It is important to note that reuse of developed components may adversely affect the security of Exploiting Platforms; the use of a developed component (or fragments thereof) on both secret and lesser classified platforms has the potential to compromise the secret one. This has not been seen as a reason to increase the indicative classification.

A.4.2.5 Context

Unless stated otherwise, the following context has been assumed:

- The component is primarily within a system belonging to an aircraft system, which may be crewed or uncrewed, and may be composed of different nodes in different locations, including ground-based elements.
- The air system is military and will be performing SNEO missions.

The Security Considerations for individual components have included additional component-specific context in the rationale where necessary.

A.4.2.6 Security Attributes

The traditional purpose of accreditation was to secure government classified information from breaches in confidentiality, integrity and availability. However, for military systems, accreditation needs to be cognisant of the effect on safety and operational effectiveness, together with operational advantage, freedom of operation and reputation, and not just the classification of its data and functionality. In support of this, the Exploiting Platform may also need to record and keep track of the flow of information or activities initiated by an authorised operator or the system itself. The PRA's security functions are intended to support the Exploiting Programme in the management of the following security attributes:

- **Confidentiality** - the property that information is not made available or disclosed to unauthorised individuals, entities, or processes. Ref. [30]
- **Integrity** - the property of accuracy and completeness. Ref. [30]
- **Availability** - the property of being accessible and usable upon demand by an authorised entity. Ref. [30]
- **Accountability** - the property that ensures actions performed by an entity may be traced uniquely to that entity. Ref. [46]

- **Auditability** - the ability to obtain audit evidence and evaluate it objectively to determine the extent to which the audit criteria are fulfilled.
- **Authenticity** - the property that an entity is what it claims to be. Ref. [30]
- **Non-repudiation** - the ability to prove the occurrence of a claimed event or action and its originating entities. Ref. [30]
- **Safety related protection** - security provided for safety related functions.

Protecting confidentiality, integrity and availability remains at the heart of ensuring the effectiveness of the product, which in turn maintains the operational advantage provided to the operator. Being able to reliably audit the actions carried out using the platform, knowing that the actions are authentic and non-repudiable will show where actions have been conducted in good faith and within the applicable rules of the air or engagement, thus protecting the operator from undue economic or reputational damage.

A.4.2.6.1 Safety Related Protection

The System Integrator of an Exploiting Platform has a responsibility to ensure that any safety related functionality is protected from cyber attack. Therefore, those components considered to be safety related may need specific security protection in accordance with DO-326A Ref. [15]/DO-356A Ref. [45] or another such acceptable means of compliance.

The approach specified in DO-326A is intended to cover airworthiness-related security, and the level of this security assurance should be commensurate with the risk associated with failure. Where other security analysis is required for the system there should be a coherent approach to reduce unnecessary overheads.

The Safety and Security Considerations of each component indicate where this will probably be needed, although this will depend on analysis performed by the Exploiting Programme. The Safety Considerations have been provided in line with the Safety Analysis PYRAMID concept.

A.4.2.7 Security Functions

The PRA is a framework for software within the application layer, however, most security functionality for the Exploiting Programme will be provided by the infrastructure; these infrastructure-based controls will be reliant on the provision of necessary configuration and metadata by the components, e.g. for establishing trust anchors, identification of data sources and maintaining separation. Additionally, some security functionality is specified within the PRA that can be tailored and configured depending on the requirements of the system. All components have been assessed as to how they might be expected to contribute to security; their Security Considerations will state whether there are:

- No anticipated security related functions.
- Security related functions (SRF) that support the security activities within the system but are not directly involved in enforcing the separation of security boundaries or preventing cyber attacks. Failure of an SRF will not directly lead to a security breach but may diminish the system’s ability to detect or counter a threat (e.g. through security event logging).

- Security enforcing functions (SEF) that the component might be expected to satisfy. Security enforcing functions are specific controls that provide protection to the system (e.g. providing cryptography). Failure of an SEF could lead to a security breach.

This is a broader classification than provided by the Common Criteria Evaluation Assurance Levels (EALs), see Ref. [40]. As with EALs, the level of rigour required for each component will increase with the security functions included, but will ultimately depend on the security requirements of the Exploiting Programme. However by documenting the functions and level of rigour using the same principles as Common Criteria, the functions could support the accreditation case.

These security functions support the concepts of a ‘defence in depth’ approach, with the components working with the infrastructure and any security controls external to the system to identify, protect, detect, respond, and recover from a cyber attack. The Cyber Defence PYRAMID concept provides a framework for how the PRA helps address cyber-related concerns.

A.4.2.7.1 Security Related Functions

The security related functions identified in the components include the following:

- **Back-up and Recovery**, supporting data archiving and the ability to recover data from that archive in the event of data loss.
- **Classification of Data**, supporting the handling of data, cross-domain separation and continued confidentiality.
- **Identifying Data Sources**, supporting trust in the system and authenticity.
- **Logging of Security Data**, supporting forensic investigations into possible security breaches and cyber attacks, etc. See the Recording and Logging PYRAMID concept for the types of data expected to be logged.
- **Maintaining Audit Records**, supporting audit, non-repudiation and accountability of mission data and events.
- **Supporting Safe Operation** and the continued airworthiness of the system.
- **Supporting Secure Remote Operation** of an uncrewed platform.
- **System Status and Monitoring**, supporting identification of unexpected activity and therefore possible security breaches and cyber attacks.
- **Warnings and Notifications**, supporting awareness of unexpected activity and therefore possible security breaches and cyber attacks.

A.4.2.7.2 Security Enforcing Functions

The security enforcing functions identified in the components include the following:

- **Applying Emission Control Rules**, providing functions to restrict communications with outside entities.
- **Detecting Security Breaches**, providing functions to identify a security breach or cyber attack.
- **Encrypting Data**, providing functions to keep data confidential.

- **Ensuring Separation of Security Domains**, providing functions that prevent unintentional cross-domain communication.
- **HW Authentication**, providing a basis of trust in the underlying hardware.
- **Preventing Cyber Attacks and Malware**, providing protection from the effects of malicious cyber activity.
- **Protecting Integrity of Data**, providing functions to maintain data accuracy and completeness of data.
- **Rendering Sensitive Data Inaccessible**, providing functions to enable sanitisation of data.
- **Restricting Access to Data**, providing functions that ensure data access is restricted to allowed system elements or operators with appropriate clearance and a need to know.
- **Securing Communications**, providing functions to ensure confidentiality of data and voice communications.
- **User Login and Authentication**, providing functions to manage access to the system.
- **Verifying Integrity of Data**, providing functions that confirm data is accurate and complete.
- **Verifying Integrity of Software**, providing a basis of trust in the software within the system.

A.4.2.8 Security Domains

As the Exploiting Programme will have its own security concerns and architecture, the indicative classification provided is meant only as a guide to support security domain segregation, it is not a requirement on an Exploiting Programme.

To support the segregation of classified information and functionality within the Exploiting Programme, the following classifications are used:

- Official (O).
- Official Sensitive (O-S).
- Secret Coalition Eyes Only (SCEO).
- Secret National Eyes Only (SNEO).
- Top Secret (TS).

Note: Sensitive and Coalition or National Eyes Only are handling conditions and not classifications in their own right. The terms “Coalition” and “National” are generic placeholders that should be replaced by the appropriate terms when applied.

This classification within the component's Security Considerations is only a guide; the classification of the implemented component and other requirements for segregation will depend on the operational context of the Exploiting Programme and define its security architecture. The implementer is free to choose their own classifications and partition the components as they see fit, with some components being instantiated within several security classification domains, with communication between domains being limited to specifically defined and authorised communications channels with appropriate security enforcing functions to monitor and verify inter-domain traffic.

Within the PRA, as an indicative guide, each component has been defined as having one or more of these security classifications. The classification of the Platform Specific Model (PSM) component and other requirements for segregation will depend on the operational context of the Exploiting Programme and define its security architecture. The security domain that the component is in will depend on:

- The classification of the data the component is processing.
- The classification of algorithms within the component.
- The required integrity of the data processed by the component.
- Security protection requirements of safety-related functions.
- Any intellectual property concerns.

Due to intellectual property, the potential for invalidating certification, and international treaty considerations, there may be elements of the system that although accessible to the developer are not accessible to the operator or to potential adversaries. This level of protection needs to be included when considering the security domains.

A.4.2.9 Security Approach Concept Summary

The following table provides a summary of this PYRAMID Concept. For guidance on understanding the table layout see section, PYRAMID Concept Summary Table Guidance.

	Security Approach Concept Summary	PRA Standard Approach
General PRA component support for this PYRAMID Concept	The PRA defines a set of application layer components that may contain security related functions (SRFs) and security enforcing functions (SEFs) in support of system security. Where the need for SRFs and SEFs is identified at the application layer, these should be implemented in accordance with the PRA subject matter definitions. The PYRAMID concept does not identify any generic component behaviours directly related to security.	In principle, any PRA component can provide a SRF or SEF, so long as the function falls within the scope of the PRA components subject matter. The PRA does not mandate the implementation of any such functions, but each PRA component definition identifies the nature of SRFs and SEFs that the component could potentially include given its subject matter. The PRA assigns indicative security classifications to PRA components, based on an assessment of generic security risk within the context of a military security environment. These indicative classifications are not requirements on an Exploiting Programme.
Specific PRA Component support for the PYRAMID Concept	The PYRAMID concept does not identify any specialised subject matters in relation to security.	The PRA defines a number of components whose role relates to security. For more information see the Cyber Defence concept.
Exploiter considerations	The following security related considerations are a matter for individual Exploiting Programmes: • Specifying the security environment and security risks applicable to the Exploiting Programme. • How components and information should be classified. • Defining the security architecture including the specific approach to system partitioning to meet security requirements, for example, the use of component variants and instances (e.g. within multiple security domains), and the use of extensions. • Demonstrating that the security targets have been met.	The PRA does not mandate a specific approach in respect of these considerations.

Table 25: Security Approach PYRAMID Concept Summary

Appendix B: Use Cases

B.1 Introduction

This Appendix defines use cases that provide a reference to Exploiters who want to understand how a generic component, as expressed within the component composition can interact in order to deliver required functionality. Each use case is illustrated from the view point of a single generic component which does not contain a specific subject matter, and therefore provide conventional patterns of interaction that are applicable to any PRA component.

These interactions are solely based upon the use of the component services defined in the PYRAMID Technical Standard, Ref. [1], section Component Composition (i.e. not based on services on any particular PRA component), and are intended to provide an understanding to Exploiters on how the services within the Component Composition can be used.

System-level views, showing how multiple components interact to achieve mission objectives, are given in Appendix C: Interaction Views. Each individual use case has one or more sequence diagrams that give examples of typical scenarios for that use case. The sequence diagrams contained in the Determine Solution, Fulfil Requirement, Request Authorisation and Receive Authorisation Revocation use cases omit the internal interactions between services within the component. Internal interactions are only shown where they do not add unnecessary complexity to diagrams and may be helpful to the reader.

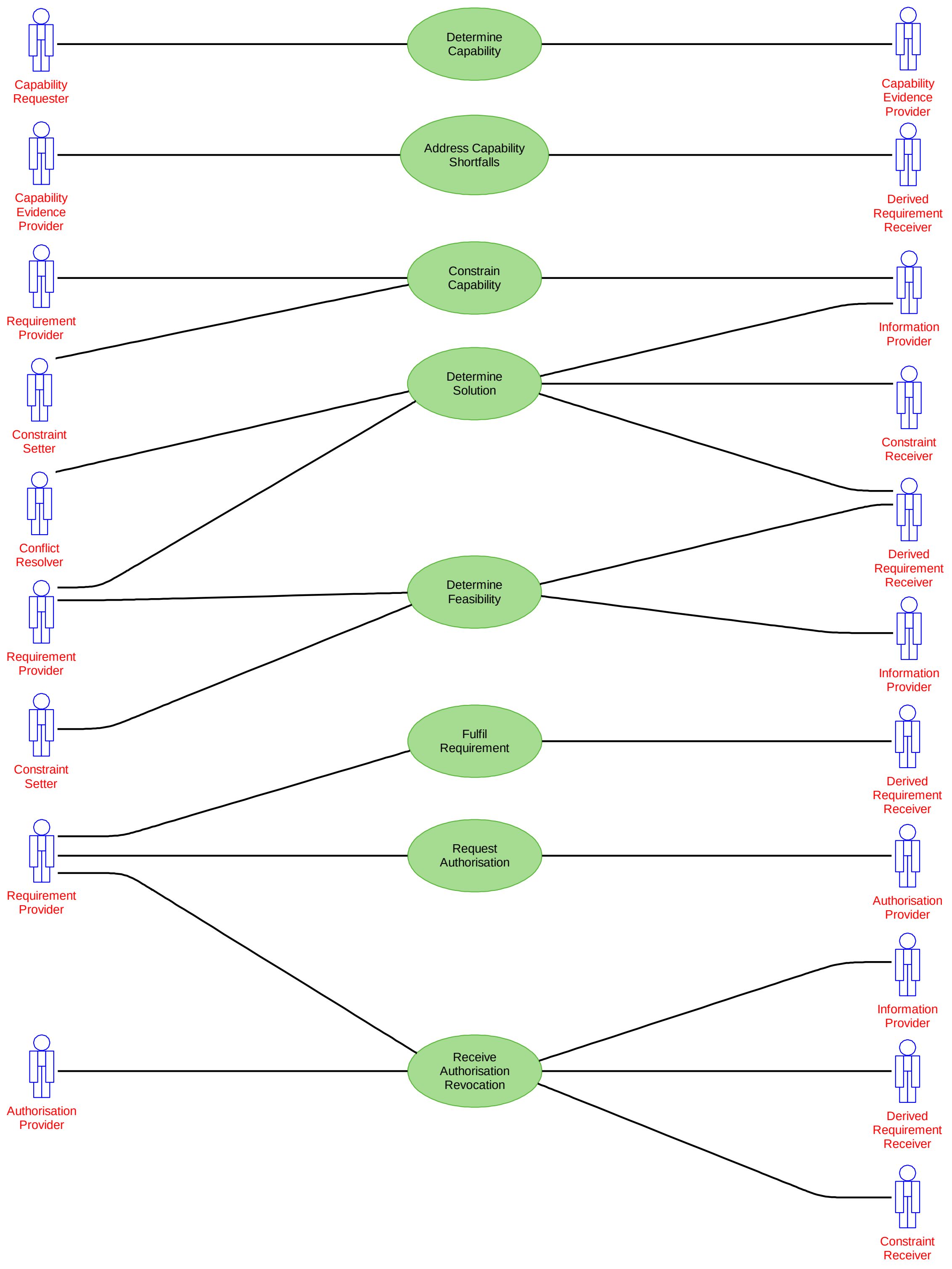


Figure 129: Use Case Summary Diagram

Figure 129: Use Case Summary Diagram summarises the use cases that the component composition supports.

B.2 Actors

An actor represents the role of a user - which may be a person, an external system or, frequently, another component - that can interact with a generic component, as expressed within the component composition. Actors are shown in Figure 129: Use Case Summary Diagram and the sequence diagrams in later sections. The actors are defined as follows:

Requirement Provider

An external actor or component that provides the requirement to be achieved, the measurement criterion by which the achievement of the requirement is assessed, and also tells the component to fulfil the requirement.

Information Provider

An external actor or component that provides information that the component needs to know in order to determine a solution.

Derived Requirement Receiver

An external actor or component that receives derived requirements (e.g. an action to be performed or a constraint on behaviour) and provides information against them.

Constraint Setter

An external actor or component that defines a Constraint, which may include information about the operating conditions or environment in which the constraint is applicable.

Capability Evidence Provider

An external actor or component that provides evidence about a capability that a component relies on. This may be the provider of a capability, such as a physical resource or a component. It could also be Anomaly Detection, Health Assessment or Cyber Defence, any of which could supply evidence for the capability of other components or parts of a system.

Capability Requester

An external actor or component that requires an understanding of the capability of a component. In the case of other components this is used to determine their own capability. In the case of an external actor, such as an operator accessing the information through the HMI, this could contribute to an understanding of the capability of a system as a whole.

Conflict Resolver

An external actor or instance of the Conflict Resolution component that manages brokerage and arbitration of requirement conflicts.

Constraint Receiver

An external actor or component that has its behaviour limited by a Constraint_Dependency in order for the component that sets the constraint to execute a solution.

Authorisation Provider

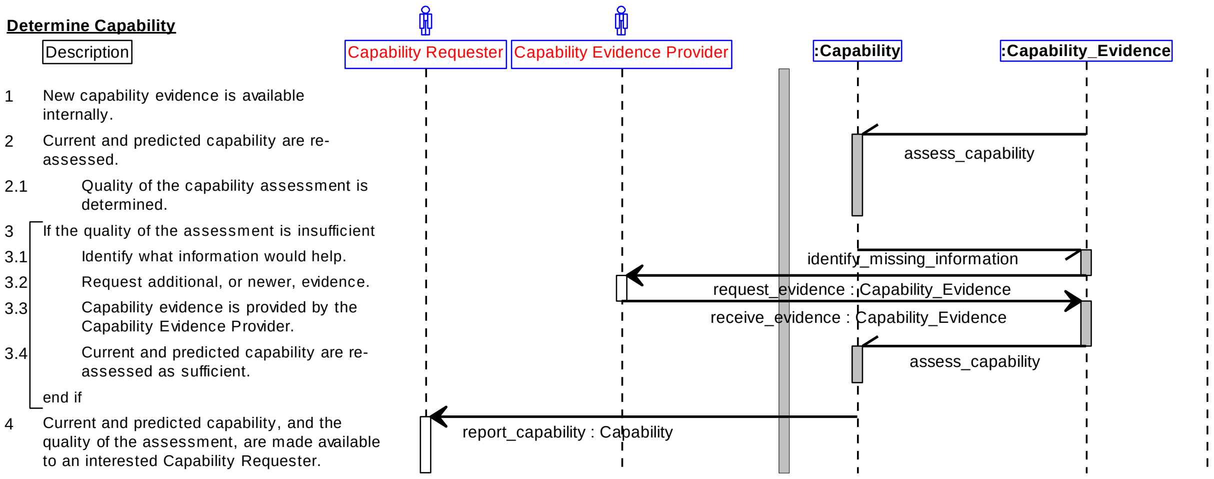
An external actor or instance of the Authorisation component that is responsible for obtaining authorisation that the component needs in order to carry out an activity.

B.3 Use Cases

B.3.1 Determine Capability

In this use case, Capability (current and predicted) is kept up-to-date as the evidence from Capability Evidence Providers changes. The three sequence diagrams show different styles of interaction between the component and a Capability Requester.

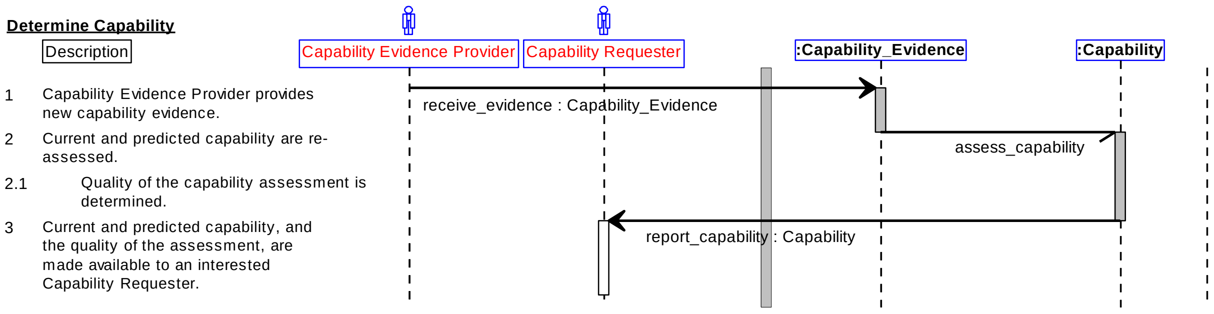
This use case shows how a component exchanges capability information with others to contribute to the Capability Management PYRAMID concept.



Determine Capability
Description
1 New capability evidence is available internally.
2 Current and predicted capability are re-assessed.
2.1 Quality of the capability assessment is determined.
3 If the quality of the assessment is insufficient
3.1 Identify what information would help.
3.2 Request additional, or newer, evidence.
3.3 Capability evidence is provided by the Capability Evidence Provider.
3.4 Current and predicted capability are re-assessed as sufficient.
end if
4 Current and predicted capability, and the quality of the assessment, are made available to an interested Capability Requester.
Capability Requester
Capability Evidence Provider
:Capability
:Capability_Evidence
assess_capability
identify_missing_information
request_evidence : Capability_Evidence
receive_evidence : Capability_Evidence
assess_capability
report_capability : Capability

Figure 130: Capability Reassessment

In the Figure 130: Capability Reassessment sequence diagram no new capability evidence is provided, but capability assessment is triggered internally, for example by the passage of time. The component's capability is re-assessed and its quality re-determined. If the quality is determined to be insufficient then further capability evidence is obtained and the capability is reassessed. The revised assessment is made available to interested Capability Requesters.



Determine Capability
Description
1 Capability Evidence Provider provides new capability evidence.
2 Current and predicted capability are re-assessed.
2.1 Quality of the capability assessment is determined.
3 Current and predicted capability, and the quality of the assessment, are made available to an interested Capability Requester.
Capability Evidence Provider
Capability Requester
:Capability_Evidence
:Capability
receive_evidence : Capability_Evidence
assess_capability
report_capability : Capability

Figure 131: Fresh Capability Evidence

In the Figure 131: Fresh Capability Evidence sequence diagram, a Capability Evidence Provider provides new capability evidence. This could be routine information, test results (possibly as a consequence of a request for missing information that was sent out earlier) or an anomaly.

The component's capability is re-assessed and its quality re-determined. The revised assessment is made available to Capability Requesters.

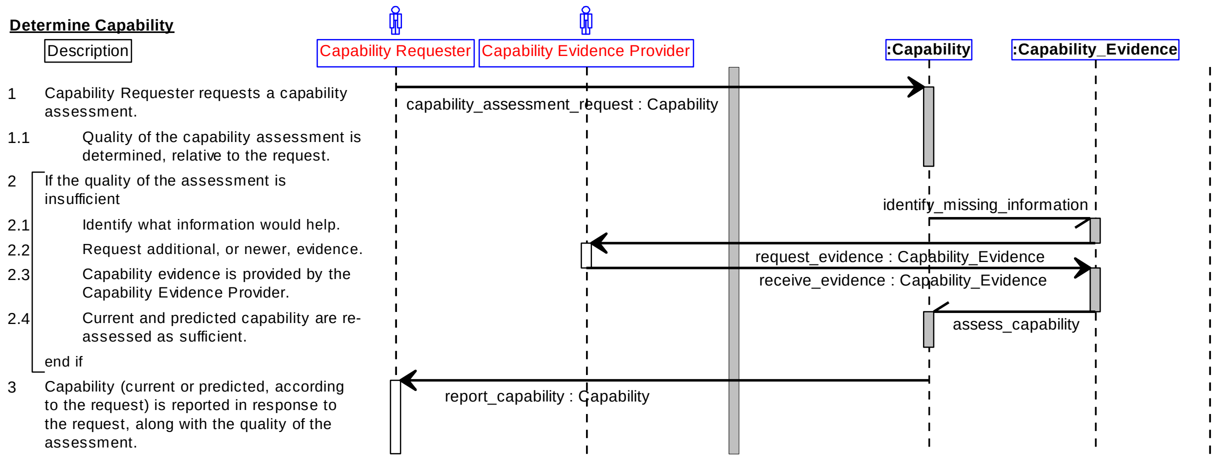

Determine Capability
Description
Capability Requester
Capability Evidence Provider
:Capability
:Capability_Evidence
1 Capability Requester requests a capability assessment.
1.1 Quality of the capability assessment is determined, relative to the request.
2 If the quality of the assessment is insufficient
2.1 Identify what information would help.
2.2 Request additional, or newer, evidence.
2.3 Capability evidence is provided by the Capability Evidence Provider.
2.4 Current and predicted capability are re-assessed as sufficient.
end if
3 Capability (current or predicted, according to the request) is reported in response to the request, along with the quality of the assessment.
capability_assessment_request : Capability
identify_missing_information
request_evidence : Capability_Evidence
receive_evidence : Capability_Evidence
assess_capability
report_capability : Capability

Figure 132: Requested Capability Assessment

In the Figure 132: Requested Capability Assessment sequence diagram, a Capability Requester makes an explicit request for capability information. The component's capability is assessed and if the quality is inadequate, the component requests additional information from a Capability Evidence Provider, that could improve the quality, and the capability is reassessed. The assessment is then made available to interested Capability Requesters.

B.3.2 Address Capability Shortfalls

In this use case, shortfalls in capability are identified, with requirements being set in an effort to re-establish the lost or degraded capability. This use case shows how component level shortfalls are addressed in line with the Capability Management PYRAMID concept.

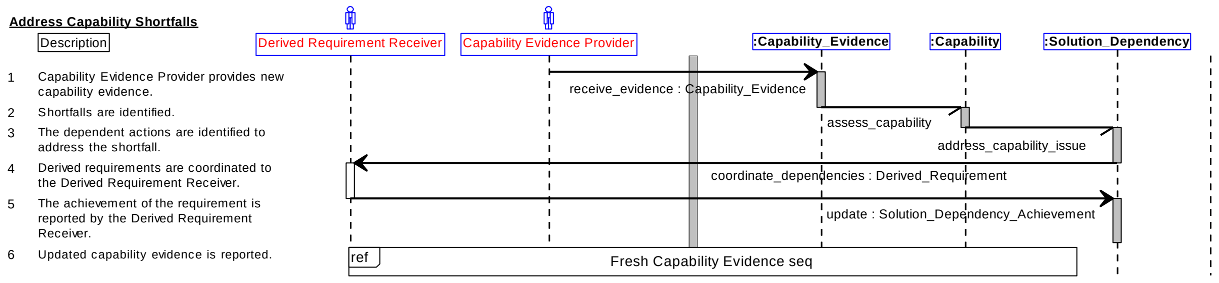

Address Capability Shortfalls
Description
Derived Requirement Receiver
Capability Evidence Provider
:Capability_Evidence
:Capability
:Solution_Dependency
1 Capability Evidence Provider provides new capability evidence.
2 Shortfalls are identified.
3 The dependent actions are identified to address the shortfall.
4 Derived requirements are coordinated to the Derived Requirement Receiver.
5 The achievement of the requirement is reported by the Derived Requirement Receiver.
6 Updated capability evidence is reported.
receive_evidence : Capability_Evidence
assess_capability
address_capability_issue
coordinate_dependencies : Derived_Requirement
update : Solution_Dependency_Achievement
ref
Fresh Capability Evidence seq

Figure 133: Address Capability Shortfalls

In the Figure 133: Address Capability Shortfalls sequence diagram, the Capability Evidence Provider provides new capability evidence to the component. The component assesses this evidence to identify any shortfalls. Dependent activities are then identified to address these shortfalls. The component provides derived requirements to the Derived Requirement Receiver which in turn reports back that these derived requirements have been achieved to the component. The updated capability evidence is then reported as shown in Figure 132: Fresh Capability Evidence.

B.3.3 Constrain Capability

In this use case, the capability of the component to do something is constrained by an external source. This use case shows how a component contributes to the Constraint Management PYRAMID concept.

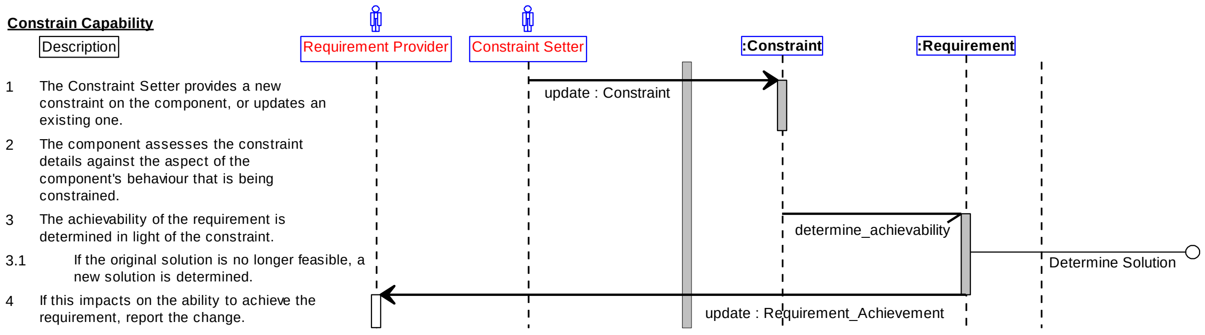

Constrain Capability
Description
1 The Constraint Setter provides a new constraint on the component, or updates an existing one.
2 The component assesses the constraint details against the aspect of the component's behaviour that is being constrained.
3 The achievability of the requirement is determined in light of the constraint.
3.1 If the original solution is no longer feasible, a new solution is determined.
4 If this impacts on the ability to achieve the requirement, report the change.
Requirement Provider
Constraint Setter
:Constraint
:Requirement
update : Constraint
determine_achievability
Determine Solution
update : Requirement_Achievement

Figure 134: Apply New Constraint

In the Figure 134: Apply New Constraint sequence diagram, a Constraint Setter provides a new constraint on the component, or updates an existing one, which then constrains the component's behaviour. The feasibility of the currently in progress solution is checked (see the Determine Feasibility use case), and if determined to be no longer achievable a new solution is determined (see the Determine Solution use case). Any changes to the achievability of the requirement are also determined and reported to the Requirement Provider if necessary.

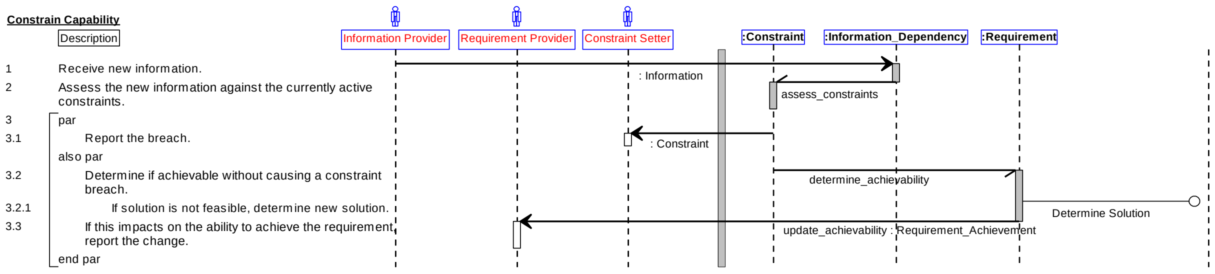

Constrain Capability
Description
1 Receive new information.
2 Assess the new information against the currently active constraints.
3 par
3.1 Report the breach.
also par
3.2 Determine if achievable without causing a constraint breach.
3.2.1 If solution is not feasible, determine new solution.
3.3 If this impacts on the ability to achieve the requirement report the change.
end par
Information Provider
Requirement Provider
Constraint Setter
:Constraint
:Information_Dependency
:Requirement
: Information
assess_constraints
: Constraint
determine_achievability
Determine Solution
update_achievability : Requirement_Achievement

Figure 135: Constrain Capability

In the Figure 135: Constrain Capability sequence diagram, an Information Provider provides new information to an active component. The component assesses this new information against the currently active constraints to see if any constraint is breached. As a constraint is breached this is reported to the Constraint Setter. The solution is checked, and if it is determined to be causing the constraint to be breached or is incompatible with the activities necessary to remove the breach condition then the solution is modified accordingly (see the Determine Solution use case). If a modified solution is not feasible then the existing solution ceases to be used and the requirement is reported as being unachievable.

B.3.4 Determine Solution

In this use case a solution to achieve a requirement is planned, taking into account capability, constraints, resource availability and associated measurement criteria. An indication of whether the requirement is achievable is reported along with the cost, which is measured using metrics as defined by the measurement criteria. Issues (such as resource conflicts) which cannot be resolved within the scope of the component's subject matter are also reported.

This use case shows the part a single component plays in finding a solution to a mission objective, as described in the Control Architecture and Dependency Management PYRAMID concepts.

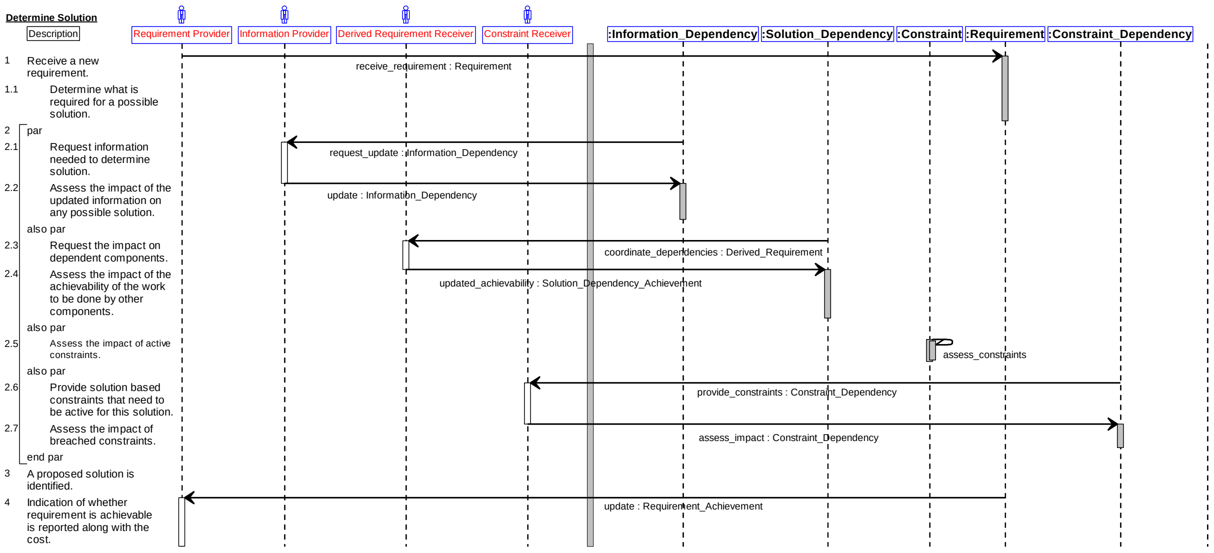

Determine Solution
Description
Requirement Provider
Information Provider
Derived Requirement Receiver
Constraint Receiver
:Information_Dependency
:Solution_Dependency
:Constraint
:Requirement
:Constraint_Dependency
1 Receive a new requirement.
1.1 Determine what is required for a possible solution.
2 par
2.1 Request information needed to determine solution.
2.2 Assess the impact of the updated information on any possible solution.
also par
2.3 Request the impact on dependent components.
2.4 Assess the impact of the achievability of the work to be done by other components.
also par
2.5 Assess the impact of active constraints.
also par
2.6 Provide solution based constraints that need to be active for this solution.
2.7 Assess the impact of breached constraints.
end par
3 A proposed solution is identified.
4 Indication of whether requirement is achievable is reported along with the cost.
receive_requirement : Requirement
request_update : Information_Dependency
update : Information_Dependency
coordinate_dependencies : Derived_Requirement
updated_achievability : Solution_Dependency_Achievement
assess_constraints
provide_constraints : Constraint_Dependency
assess_impact : Constraint_Dependency
update : Requirement_Achievement

Figure 136: New Requirement Trigger

In the Figure 136: New Requirement Trigger sequence diagram, a Requirement Provider provides a new requirement. To determine the solution, the component requests information from the Information Provider, assesses the impact of any active constraints, determines the solution dependencies that need to be achieved by the Derived Requirement Receiver, and determines solution based constraints that need to be adhered to.

The component will place the solution dependencies and constraints on relevant components and based upon feedback ensure that these can be met. The achievability and achievement of the solution dependencies, expressed in terms of requirements, are reported to the component.

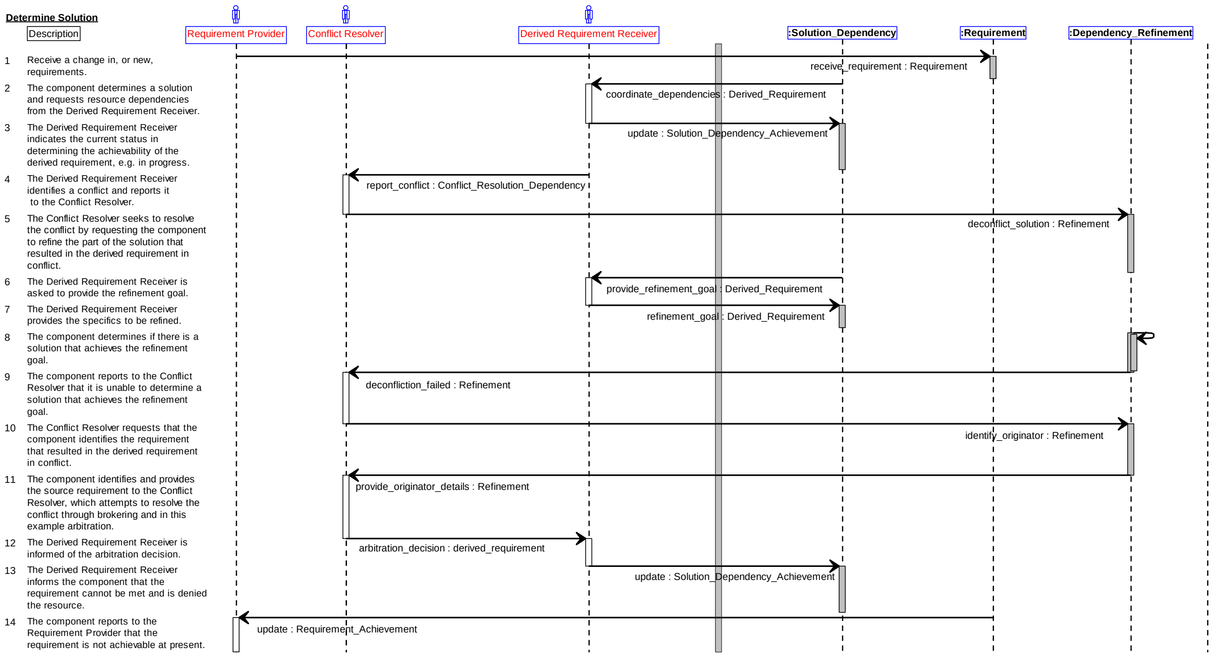

Determine Solution
Description
Requirement Provider
Conflict Resolver
Derived Requirement Receiver
:Solution_Dependency
:Requirement
:Dependency_Refinement
1 Receive a change in, or new, requirements.
2 The component determines a solution and requests resource dependencies from the Derived Requirement Receiver.
3 The Derived Requirement Receiver indicates the current status in determining the achievability of the derived requirement, e.g. in progress.
4 The Derived Requirement Receiver identifies a conflict and reports it to the Conflict Resolver.
5 The Conflict Resolver seeks to resolve the conflict by requesting the component to refine the part of the solution that resulted in the derived requirement in conflict.
6 The Derived Requirement Receiver is asked to provide the refinement goal.
7 The Derived Requirement Receiver provides the specifics to be refined.
8 The component determines if there is a solution that achieves the refinement goal.
9 The component reports to the Conflict Resolver that it is unable to determine a solution that achieves the refinement goal.
10 The Conflict Resolver requests that the component identifies the requirement that resulted in the derived requirement in conflict.
11 The component identifies and provides the source requirement to the Conflict Resolver, which attempts to resolve the conflict through brokering and in this example arbitration.
12 The Derived Requirement Receiver is informed of the arbitration decision.
13 The Derived Requirement Receiver informs the component that the requirement cannot be met and is denied the resource.
14 The component reports to the Requirement Provider that the requirement is not achievable at present.
receive_requirement : Requirement
coordinate_dependencies : Derived_Requirement
update : Solution_Dependency_Achievement
report_conflict : Conflict_Resolution_Dependency
deconflict_solution : Refinement
provide_refinement_goal : Derived_Requirement
refinement_goal : Derived_Requirement
deconfliction_failed : Refinement
identify_originator : Refinement
provide_originator_details : Refinement
arbitration_decision : derived_requirement
update : Solution_Dependency_Achievement
update : Requirement_Achievement

Figure 137: Report Resource Conflict

The Figure 137: Report Resource Conflict sequence diagram shows actor to actor interactions, which would not normally be shown, in order to provide greater context for the interactions with the component.

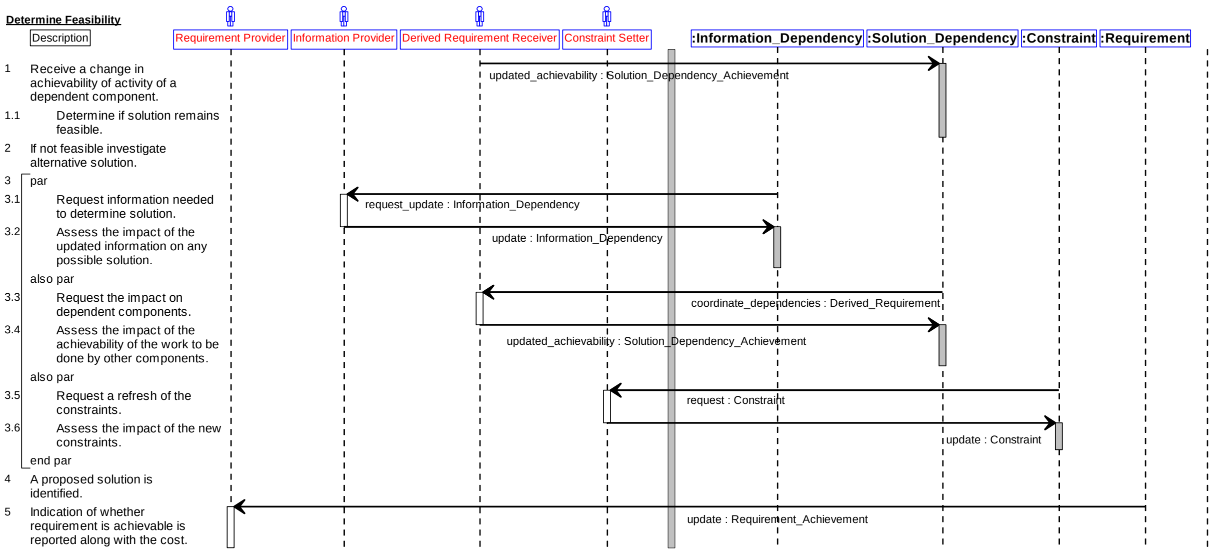
In the Figure 137: Report Resource Conflict sequence diagram, a change to an existing requirement, or a new requirement, is received. One of the dependencies requires the use of a resource that the Derived Requirement Receiver, as Resource Provider, is unable to allocate. The Conflict Resolver attempts to resolve the conflict by requesting a change to the solution resulting in the derived requirements, but no suitable alternative solution is available that would allow the derived requirements to be changed in such a way. The Conflict Resolver requests the identity of the requirement resulting in the derived requirements to support conflict resolution by brokerage and arbitration, which results in the requested resource allocation being denied. The component reports that the requirement is not currently achievable to the Requirement Provider.

This sequence diagram shows the steps relevant to reporting a resource conflict, but the same process can be used to broker and arbitrate conflicts of all kinds, not just resource conflicts.

B.3.5 Determine Feasibility

In this use case, the feasibility of carrying out the solution intended to achieve a requirement is determined. This includes solutions planned to be carried out in the future as well as those being carried out now.

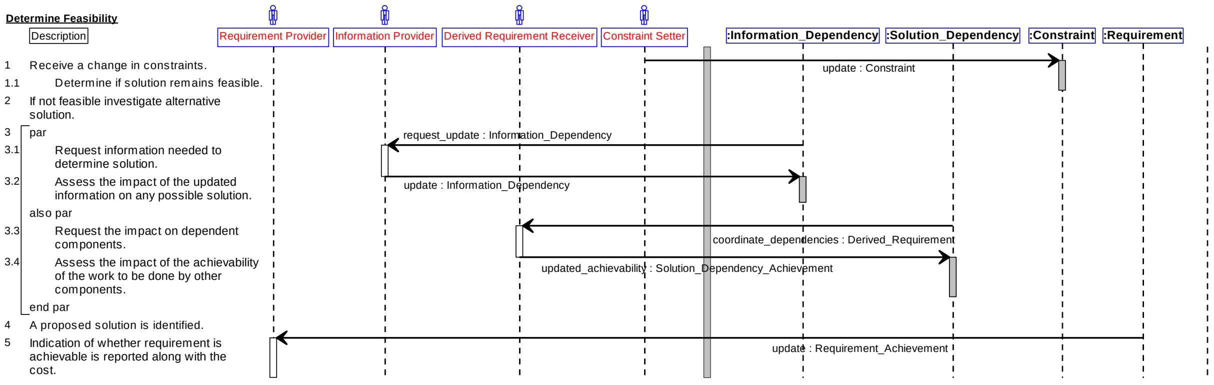
This use case shows how each component monitors the feasibility of a planned solution when circumstances change, as discussed in the Dependency Management PYRAMID concept, and shows the process by which an alternative solution may be identified following the determination that the current solution is not feasible.


Determine Feasibility
Description
Requirement Provider
Information Provider
Derived Requirement Receiver
Constraint Setter
:Information_Dependency
:Solution_Dependency
:Constraint
:Requirement
updated_achievability : Solution_Dependency_Achievement
request_update : Information_Dependency
update : Information_Dependency
coordinate_dependencies : Derived_Requirement
updated_achievability : Solution_Dependency_Achievement
request : Constraint
update : Constraint
update : Requirement_Achievement

1 Receive a change in achievability of activity of a dependent component.
1.1 Determine if solution remains feasible.
2 If not feasible investigate alternative solution.
3 par
3.1 Request information needed to determine solution.
3.2 Assess the impact of the updated information on any possible solution.
also par
3.3 Request the impact on dependent components.
3.4 Assess the impact of the achievability of the work to be done by other components.
also par
3.5 Request a refresh of the constraints.
3.6 Assess the impact of the new constraints.
end par
4 A proposed solution is identified.
5 Indication of whether requirement is achievable is reported along with the cost.

Figure 138: Change in the Achievability of Dependent Component Trigger

In the Figure 138: Change in the Achievability of Dependent Component Trigger sequence diagram, a Derived Requirement Receiver provides an achievability update on its activities. The solution feasibility is determined and a new solution is investigated if the original is no longer feasible. To determine the new solution, the component requests information from the Information Provider, solution dependencies to be achieved from the Derived Requirement Receiver and a refresh of constraints from the Constraint Setter, before identifying a preferred solution.



Determine Feasibility
Description
Requirement Provider
Information Provider
Derived Requirement Receiver
Constraint Setter
:Information_Dependency
:Solution_Dependency
:Constraint
:Requirement
1 Receive a change in constraints.
1.1 Determine if solution remains feasible.
2 If not feasible investigate alternative solution.
3 par
3.1 Request information needed to determine solution.
3.2 Assess the impact of the updated information on any possible solution.
also par
3.3 Request the impact on dependent components.
3.4 Assess the impact of the achievability of the work to be done by other components.
end par
4 A proposed solution is identified.
5 Indication of whether requirement is achievable is reported along with the cost.
update : Constraint
request_update : Information_Dependency
update : Information_Dependency
coordinate_dependencies : Derived_Requirement
updated_achievability : Solution_Dependency_Achievement
update : Requirement_Achievement

Figure 139: Change in the Constraints Trigger

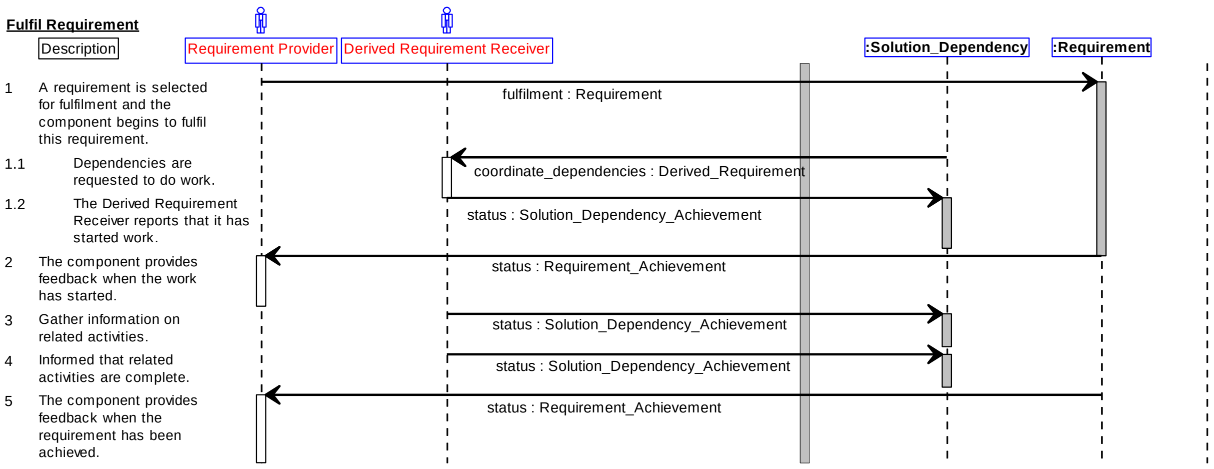
In the Figure 139: Change in the Constraints Trigger sequence diagram, the Constraint Setter provides a change in constraints. The solution feasibility is determined and a new solution is investigated if the original is no longer feasible. To determine the new solution, the component requests information from the Information Provider, and solution dependencies to be achieved from the Derived Requirement Receiver, before identifying a preferred solution.

B.3.6 Fulfil Requirement

In this use case a requirement is fulfilled by executing the planned solution; achievement of the requirement is assessed and reported when requested by the Requirement Provider, at regular intervals, or when requirements have been met.

The use case assumes that the planned solution still meets the requirements. The Determine Feasibility use case shows how a solution replan is triggered if necessary.

This use case shows the part played by a single component in carrying out a mission objective, as described in the Control Architecture and Dependency Management PYRAMID concepts.



Fulfil Requirement
Description
Requirement Provider
Derived Requirement Receiver
:Solution_Dependency
:Requirement
1 A requirement is selected for fulfilment and the component begins to fulfil this requirement.
1.1 Dependencies are requested to do work.
1.2 The Derived Requirement Receiver reports that it has started work.
2 The component provides feedback when the work has started.
3 Gather information on related activities.
4 Informed that related activities are complete.
5 The component provides feedback when the requirement has been achieved.
fulfilment : Requirement
coordinate_dependencies : Derived_Requirement
status : Solution_Dependency_Achievement
status : Requirement_Achievement
status : Solution_Dependency_Achievement
status : Solution_Dependency_Achievement
status : Requirement_Achievement

Figure 140: Fulfil Requirement

In the Figure 140: Fulfil Requirement sequence diagram, a Requirement Provider selects a previously defined requirement to be fulfilled. The component requests work from dependencies and then informs the Requirement Provider when the work has begun. Information is then gathered on the progress of dependency activities and the component's own activities, and reported once the requirement has been achieved.

B.3.7 Request Authorisation

In this use case, a component determines a solution that requires authorisation in order to enact. In order to progress this solution, the component first requests the achievability of the authorisation from the Authorisation Provider. Once the feasibility of the solution is known, the component requests the authorisation to enact the solution.

This use case shows how a component can request the required authorisation in a way which is consistent with the Autonomy PYRAMID concept.

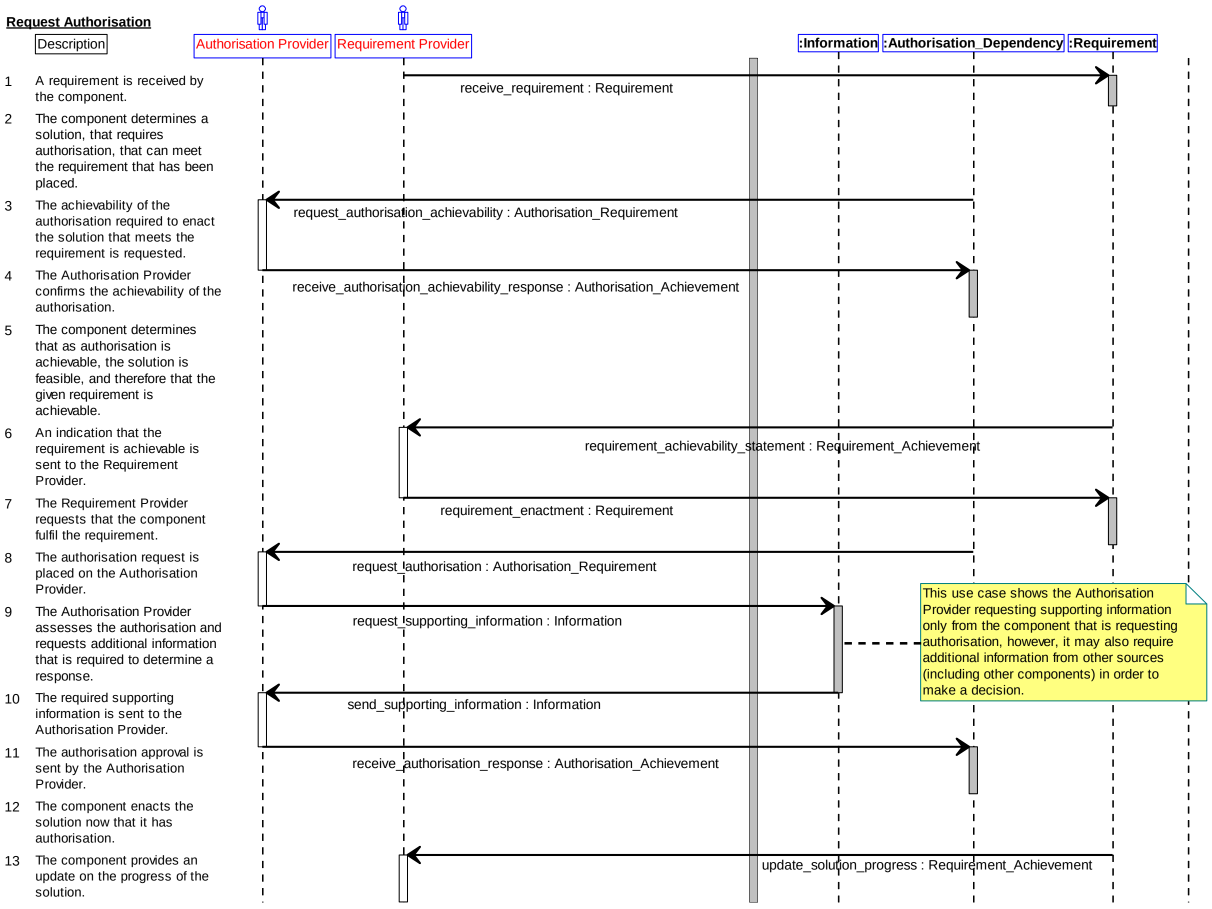

Request Authorisation
Description
Authorisation Provider
Requirement Provider
:Information
:Authorisation_Dependency
:Requirement
1 A requirement is received by the component.
2 The component determines a solution, that requires authorisation, that can meet the requirement that has been placed.
3 The achievability of the authorisation required to enact the solution that meets the requirement is requested.
4 The Authorisation Provider confirms the achievability of the authorisation.
5 The component determines that as authorisation is achievable, the solution is feasible, and therefore that the given requirement is achievable.
6 An indication that the requirement is achievable is sent to the Requirement Provider.
7 The Requirement Provider requests that the component fulfil the requirement.
8 The authorisation request is placed on the Authorisation Provider.
9 The Authorisation Provider assesses the authorisation and requests additional information that is required to determine a response.
10 The required supporting information is sent to the Authorisation Provider.
11 The authorisation approval is sent by the Authorisation Provider.
12 The component enacts the solution now that it has authorisation.
13 The component provides an update on the progress of the solution.
receive_requirement : Requirement
request_authorisation_achievability : Authorisation_Requirement
receive_authorisation_achievability_response : Authorisation_Achievement
requirement_achievability_statement : Requirement_Achievement
requirement_enactment : Requirement
request_authorisation : Authorisation_Requirement
request_supporting_information : Information
This use case shows the Authorisation Provider requesting supporting information only from the component that is requesting authorisation, however, it may also require additional information from other sources (including other components) in order to make a decision.
send_supporting_information : Information
receive_authorisation_response : Authorisation_Achievement
update_solution_progress : Requirement_Achievement

Figure 141: Request Authorisation to Progress a Solution

In the Figure 141: Request Authorisation to Progress a Solution sequence diagram, the component creates a solution to a requirement that needs authorisation, then determines that the solution is feasible based on the achievability of the authorisation. Once the solution is requested to be enacted, authorisation is obtained.

The component receives a requirement to meet, a solution that meets this is determined (refer to the Determine Solution use case for more information) and it is identified that this solution will require authorisation to enact. The component requests the achievability of the authorisation from the

Authorisation Provider. The achievability of the authorisation is confirmed by the Authorisation Provider.

The component sends confirmation of its ability to meet the requirement to the Requirement Provider. The Requirement Provider decides for this requirement to enact the requirement right away and a request is sent to the component (in other cases the Requirement Provider may request that the component first prepare to enact by obtaining authorisation prior to the enactment request so that the component is completely ready when the request is made).

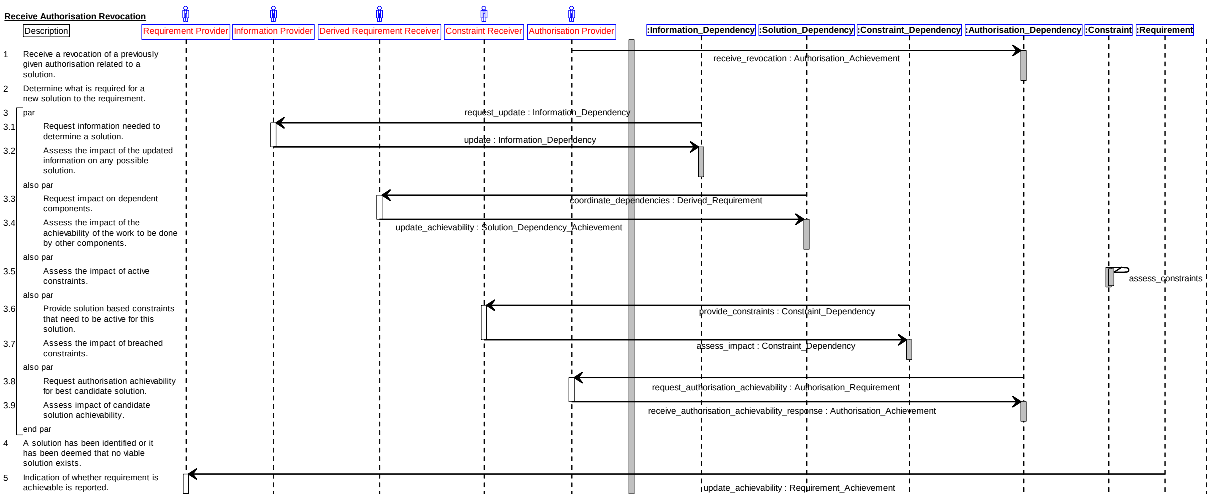
The component requests authorisation to enact this solution from the Authorisation Provider. The Authorisation Provider requires additional supporting information, and requests and receives this information from the component. The authorisation request is approved by the Authorisation Provider, making use of the supporting information received. The authorisation is received by the component. The solution is enacted by the component (refer to the Figure 140: Fulfil Requirement sequence diagram for more information) and updates regarding the progress of the solution are sent to the Requirement Provider.

B.3.8 Receive Authorisation Revocation

In this use case a previously given authorisation for a solution is revoked.

This use case shows the steps taken by a component in the attempt to still meet a requirement in the circumstance that a previously given authorisation is revoked.

Once the determination has been made as to what still needs to be done to meet the requirement, a new solution can then be determined as shown in Figure 136: New Requirement Trigger.


Receive Authorisation Revocation
Description
Requirement Provider
Information Provider
Derived Requirement Receiver
Constraint Receiver
Authorisation Provider
:Information_Dependency
:Solution_Dependency
:Constraint_Dependency
:Authorisation_Dependency
:Constraint
:Requirement
receive_revocation : Authorisation_Achievement
request_update : Information_Dependency
update : Information_Dependency
coordinate_dependencies : Derived_Requirement
update_achievability : Solution_Dependency_Achievement
assess_constraints
provide_constraints : Constraint_Dependency
assess_impact : Constraint_Dependency
request_authorisation_achievability : Authorisation_Requirement
receive_authorisation_achievability_response : Authorisation_Achievement
update_achievability : Requirement_Achievement

1 Receive a revocation of a previously given authorisation related to a solution.
2 Determine what is required for a new solution to the requirement.
3 par
3.1 Request information needed to determine a solution.
3.2 Assess the impact of the updated information on any possible solution.
also par
3.3 Request impact on dependent components.
3.4 Assess the impact of the achievability of the work to be done by other components.
also par
3.5 Assess the impact of active constraints.
also par
3.6 Provide solution based constraints that need to be active for this solution.
3.7 Assess the impact of breached constraints.
also par
3.8 Request authorisation achievability for best candidate solution.
3.9 Assess impact of candidate solution achievability.
end par
4 A solution has been identified or it has been deemed that no viable solution exists.
5 Indication of whether requirement is achievable is reported.

Figure 142: Receive Authorisation Revocation

In the Figure 142: Receive Authorisation Revocation sequence diagram, the Authorisation Provider provides a revocation to a previously given authorisation. To determine if a new solution to the requirement can be found, the component requests information from the Information Provider, assesses the impact of any active constraints, determines the solution dependencies that need to be achieved by the Derived Requirement Receiver, determines solution based constraints that need to be adhered to, and requests authorisation for the candidate solution.

The component will place the solution dependencies and constraints on relevant components and based upon feedback ensure that these can be met. The achievability of the solution dependencies, expressed in terms of requirements, as well as the response to the authorisation request, are reported to the component.

The component updates the Requirement Provider as to whether the requirement is still achievable.

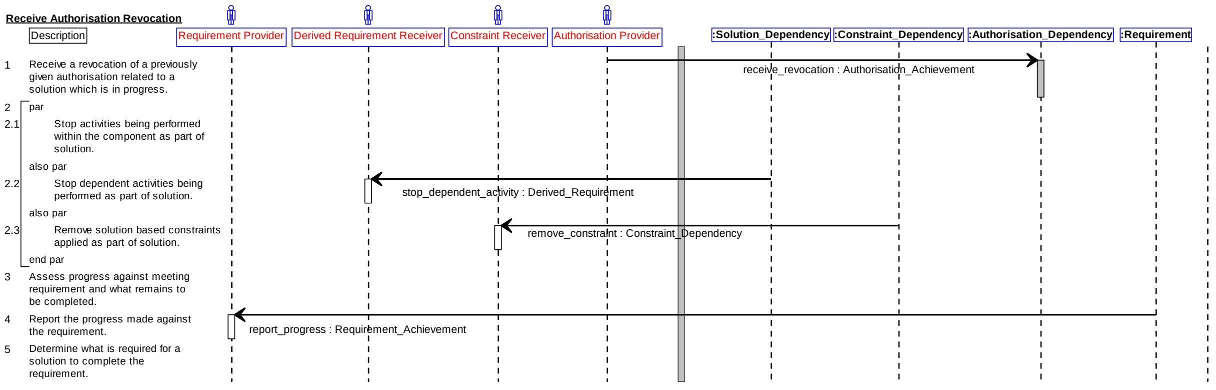

Receive Authorisation Revocation
Description
Requirement Provider
Derived Requirement Receiver
Constraint Receiver
Authorisation Provider
:Solution_Dependency
:Constraint_Dependency
:Authorisation_Dependency
:Requirement
1 Receive a revocation of a previously given authorisation related to a solution which is in progress.
2 par
2.1 Stop activities being performed within the component as part of solution.
also par
2.2 Stop dependent activities being performed as part of solution.
also par
2.3 Remove solution based constraints applied as part of solution.
end par
3 Assess progress against meeting requirement and what remains to be completed.
4 Report the progress made against the requirement.
5 Determine what is required for a solution to complete the requirement.
receive_revocation : Authorisation_Achievement
stop_dependent_activity : Derived_Requirement
remove_constraint : Constraint_Dependency
report_progress : Requirement_Achievement

Figure 143: Receive Authorisation Revocation for Ongoing Activity

In the Figure 143: Receive Authorisation Revocation for Ongoing Activity sequence diagram, the Authorisation Provider provides a revocation to a previously given authorisation for a solution that is in the process of being executed. The activities that form part of the solution are stopped, including activities being performed within the component and dependent activities in other components, and any solution based constraints applied are removed.

This sequence diagram shows the activities which form part of the solution simply being stopped. In some cases a solution cannot be simply stopped without another solution taking its place (for example a routing solution within the Routes component). In this case the component will need to begin an alternative default or backup solution, but this is not shown here.

The component determines what progress has been made against meeting the requirement and reports this to the Requirement Provider.

It is then determined what is required for a solution to complete the requirement, and a new solution can then be determined as shown in Figure 136: New Requirement Trigger.

Appendix C: Interaction Views

C.1 Introduction

This appendix describes the composition of an interaction view (IV) and contains the set of developed IVs which cover a wide range of mission system functions, as introduced in Introduction to Interaction Views.

The IVs are intended to add context to the PRA component definitions, as well as illustrating the intended use of components in PYRAMID deployments.

C.1.1 IV Scenarios

An interaction view (IV) scenario is captured as a UML/SysML use case, which contains:

- A textual description providing context for the IV scenario, including any assumptions or exclusions considered when scoping its coverage;
- The pre- and post-conditions to further define the scoped coverage of the IV scenario;
- The flow of the scenario, given by a communication diagram and descriptive text, which provides a narrative to support understanding of the IV diagram and the interactions represented on it.

An IV scenario is an event based example of how a core set of components can interact. As such it will not include provision of all of the information required to implement the described functionality. Like the scenario itself, the flow describes a specific possible implementation. Similar functionality may be achieved by using an alternate set of components or flow as determined by an Exploiter.

C.1.2 IV Diagrams

An IV contains an IV diagram (in the form of a UML communication diagram) for each use case, which illustrates how the components interact to achieve the behaviour described in the associated IV scenario(s).

Any entities described in the IV scenario that are outside the boundary of the IV diagram are represented by actors. The IV diagram event message numbers are in the order of the component interactions described in the flow. An asterisk before the name means that the event starts at the indicated number and then continues to occur from that point on. The direction of the arrow indicates where the event is enacted. An example IV diagram can be seen in Figure 144: Example Interaction View (IV).

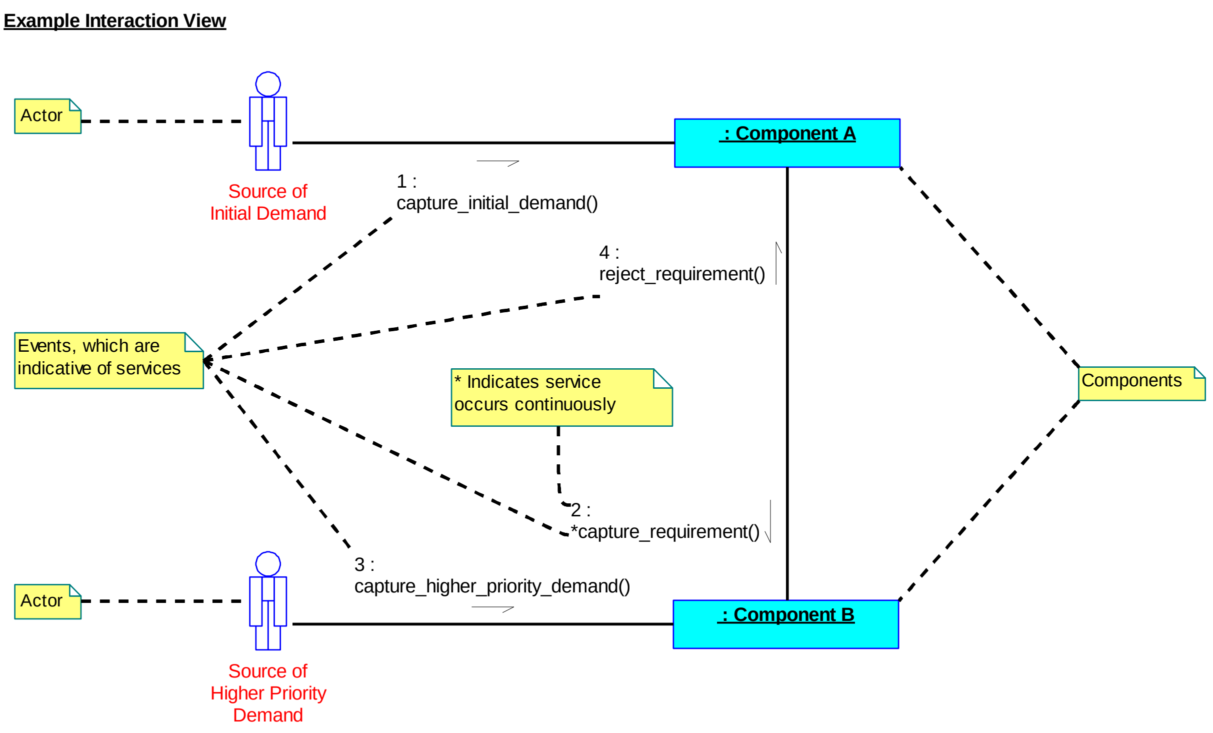

Example Interaction View
Actor
Source of Initial Demand
: Component A
1 : capture_initial_demand()
4 : reject_requirement()
Events, which are indicative of services
* Indicates service occurs continuously
Components
2 : *capture_requirement()
3 : capture_higher_priority_demand()
Actor
: Component B
Source of Higher Priority Demand

Figure 144: Example Interaction View (IV)

C.1.2.1 Components

Only the core set of components required for representing the requirements of the IV scenario are included; this is intended to reduce repetition and increase readability. For example, the Navigation IV illustrates how the Location and Orientation component interacts with a number of other components to provide positional information to the system. This positional information is required by the Vehicle Guidance component on the Vehicle Movement IV, however that IV only shows the Location and Orientation component - the other supporting components are not shown as the scope of that IV does not need to consider how the positional information is generated, only that it is.

C.1.2.2 Interactions

Associations between components indicate specific support that one component requires of another. The associations are linked to the responsibilities of the component that fulfils the requirement, and are labelled accordingly (however, these associations are only visible within the model and are not shown within this document). These responsibilities will ultimately be realised by the collaborative use of the services on the components. There may be a dialogue between the components involving multiple services: for brevity and clarity this level of detail is not shown on the diagrams.

Within the textual flow, certain behavioural patterns are used, such as:

- One component will ask other components to collaborate in determining a solution to a problem. This represents not just a simple request but an exchange of provided and required services culminating in an agreed solution.
- One component will tell another to perform a predetermined action. This represents the invocation of provided services on the receiving component to implement the determined solution to a problem.

In the Figure 144: Example Interaction View (IV), Component A continuously places requirements on Component B. Component B has a defined responsibility to determine a solution which satisfies that requirement placed on it by Component A. Upon receiving a higher priority requirement from an external actor (e.g. a manual override), Component B rejects the requirement from Component A.

C.1.2.3 Actors

Actors are defined in the context of the IV diagram they appear on. They are used to represent the role of an entity that exists outside of the IV diagram boundary (including components that are not within scope of the IV) but interacts with the components within the boundary. The actor definition includes a supporting description to clarify its role in the IV scenario.

C.2 Interaction Views

C.2.1 Vehicle Path Views

This section contains IVs relating to the path of an air vehicle and how it is controlled.

The Figure 145: Vehicle Path Views UC Diagram shows all the use cases within the Vehicle Path Views and any actors that interact with them.

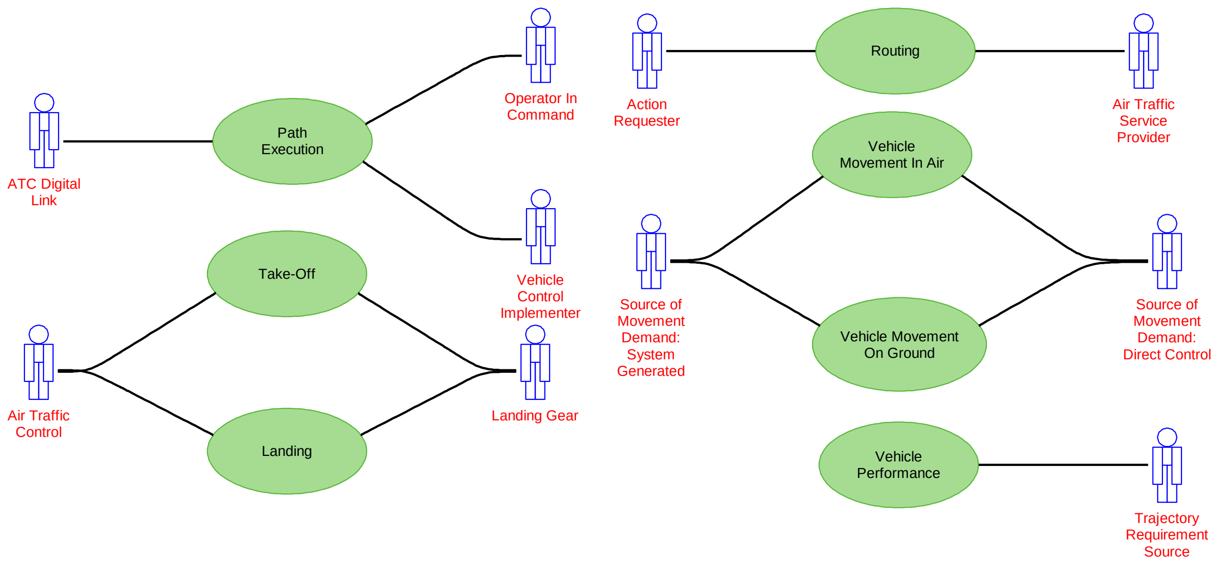

ATC Digital Link
Path Execution
Operator In Command
Vehicle Control Implementer
Take-Off
Landing
Air Traffic Control
Landing Gear
Action Requester
Routing
Air Traffic Service Provider
Vehicle Movement In Air
Source of Movement Demand: System Generated
Vehicle Movement On Ground
Source of Movement Demand: Direct Control
Vehicle Performance
Trajectory Requirement Source

Figure 145: Vehicle Path Views UC Diagram

C.2.1.1 Path Execution

This IV describes how the execution of the planned path of an air vehicle is managed over different phases of the mission, corresponding to different demands for moving the vehicle. The example uses the scenario of an air vehicle taking off, departing an airfield via a standard departure and performing a transit through controlled airspace, to demonstrate the path execution behaviour. The transit includes a search task as part of its requirements, as well as the receipt and implementation of ATC instructions.

It is assumed that:

- The system is capable of interpreting ATC instructions received via datalink and is approved to do so.

The following related areas of functionality are excluded:

- Obtaining the required clearances relating to flight in controlled airspace - obtaining authorisations.
- Coordination and control of the take-off.
- Control of the air vehicle in order to follow the determined flight path.
- Generation of routes.
- Voice communications with an ATC unit.

- Checking that the route and demanded path does not conflict with terrain, obstructions, unauthorised airspace, etc.

C.2.1.1.1 Pre-Conditions

- Airspace rules for the terminal operation area and transit airspace have been determined.
- Take-off profile requirements have been determined.
- Routing requirements for the departure and transit have been determined (e.g. appropriate SID and ATC instructions).
- Routing requirements for the search have been determined.
- Routes has generated a planned departure route.
- Routes has generated a planned transit/search route.
- Planned sequence of vehicle movement demands and their resulting flight path (take-off manoeuvre, departure route and transit/search route) has been checked.
- Take-off has been initiated.

C.2.1.1.2 View

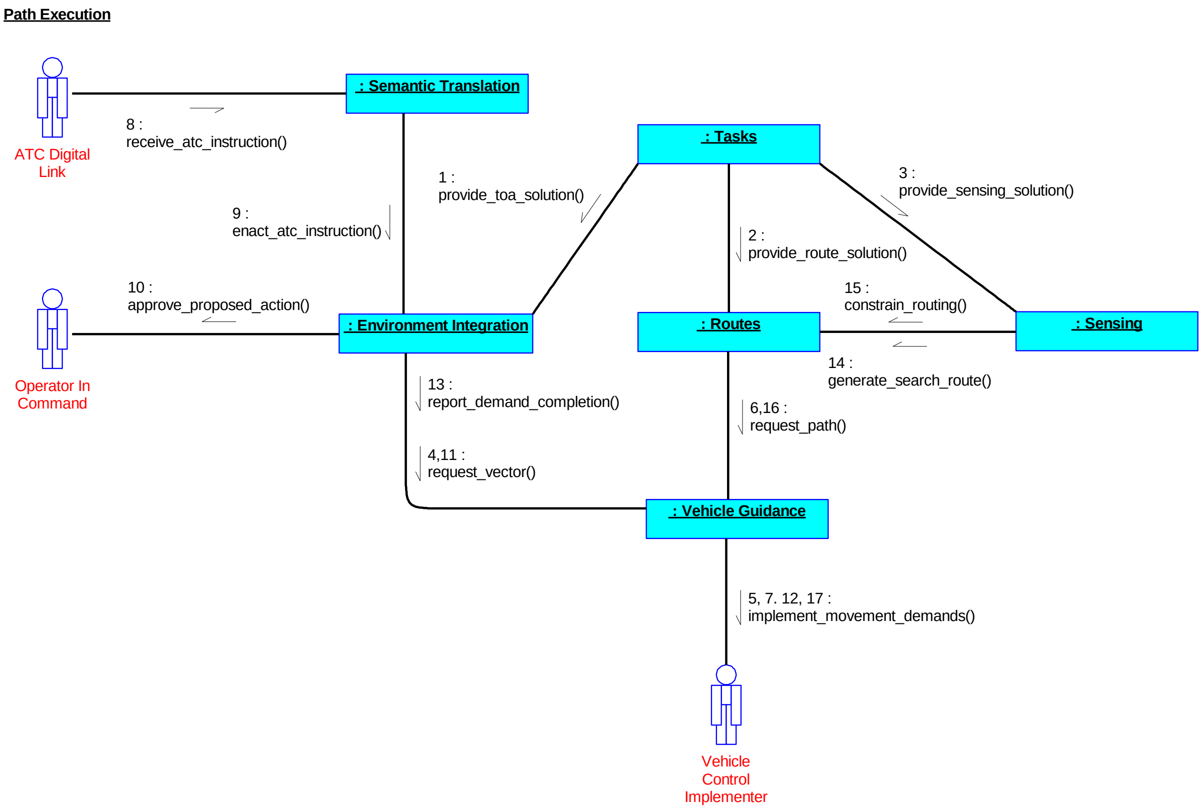

Path Execution
: Semantic Translation
ATC Digital Link
8 : receive_atc_instruction()
9 : enact_atc_instruction()
1 : provide_toa_solution()
: Tasks
3 : provide_sensing_solution()
2 : provide_route_solution()
10 : approve_proposed_action()
Operator In Command
: Environment Integration
: Routes
15 : constrain_routing()
: Sensing
14 : generate_search_route()
13 : report_demand_completion()
6,16 : request_path()
4,11 : request_vector()
: Vehicle Guidance
5, 7. 12, 17 : implement_movement_demands()
Vehicle Control Implementer

Figure 146: Path Execution IV

Tasks requests Environment Integration, Routes and Sensing to deliver action solutions relating to take-off, routing (for departure, transit and search) and sensing respectively. Tasks manages the execution of these action solutions via coordination points (e.g. between the completion of take-off and the start of the departure routing, or the start of the sensing actions at the completion point in the transit).

Vehicle Guidance receives the demand to execute the take-off profile from Environment Integration. Vehicle Guidance then generates the necessary motion commands which are executed by the Vehicle Control Implementer, the air vehicle then follows the required path, as described in the Figure 147: Take-Off IV.

On completion of the take-off profile, Vehicle Guidance receives the demand to execute the planned departure route from Routes. Vehicle Guidance then generates the necessary motion commands which are executed by the Vehicle Control Implementer. The air vehicle then follows the required path, as described in the Figure 150: Vehicle Movement In Air IV.

On completion of the departure route, Vehicle Guidance receives the demand to execute the planned transit route from Routes. Vehicle Guidance then generates the necessary motion commands which are executed by the Vehicle Control Implementer. The air vehicle then follows the required path, as described in the Figure 150: Vehicle Movement In Air IV.

ATC instructions received via datalink that affect the execution of the air vehicle transit are processed by Semantic Translation, which decodes the instructions and provides them to Environment Integration to interpret and determine any required path changes. Any such path changes will be assessed and authorised by the Operator In Command prior to their enactment by the air vehicle and will include procedural elements such as voice back-up. The Operator In Command approves the proposed action, Vehicle Guidance receives the demand for the interpreted ATC instructions commanded by Environment Integration. Implemented by the Vehicle Control Implementer, the air vehicle executes the path changes and Vehicle Guidance reports completion of the demand request.

Routes generates a planned search route, using positioning requirements provided by Sensing. Route constraints in support of the sensing actions are provided by Sensing to Routes. On completion of the transit route, Vehicle Guidance receives the demand to execute a planned search route from Routes and generates the necessary motion commands which are executed by the Vehicle Control Implementer. The air vehicle follows the provided route as described in the Figure 150: Vehicle Movement In Air IV. At the appropriate points in the search route, Sensing will activate and terminate the sensing activities.

C.2.1.1.3 Post-Conditions

- Air vehicle has completed the transit through controlled airspace.
- Air vehicle has completed the search task.

C.2.1.1.4 Actors

ATC Digital Link

A digital link to an ATC unit for two-way communication (e.g. CPDLC). This includes the elements of the system that enable such communications.

Operator In Command

An authorised operator with the command responsibility for an air vehicle.

Vehicle Control Implementer

A part of the system that implements the vehicle movement demands as movements to the control surfaces and changes to the propulsion demands.

C.2.1.2 Take-Off and Landing

C.2.1.2.1 Take-Off

This IV illustrates the process of coordinating and executing the transition of an air vehicle from ground to air. It covers the coordination of the conditions needed for take-off, performing the take-off manoeuvre and controlling the changes needed to the aircraft configuration after take-off.

It is assumed that:

- The Exploiting Platform is a conventional-take-off aircraft that takes off from a runway.
- This is a successful take-off of an air vehicle, with no need to abort.

The following related areas of functionality are excluded:

- The later stages of departure, and any interactions with Air Traffic Control that would be associated with this.
- Handover from local control to central control for the purposes of take-off.

C.2.1.2.1.1 Pre-Conditions

- The air vehicle is powered up and at the end of the runway, ready to take off.
- The air vehicle has received an operator instruction to take off.

C.2.1.2.1.2 View

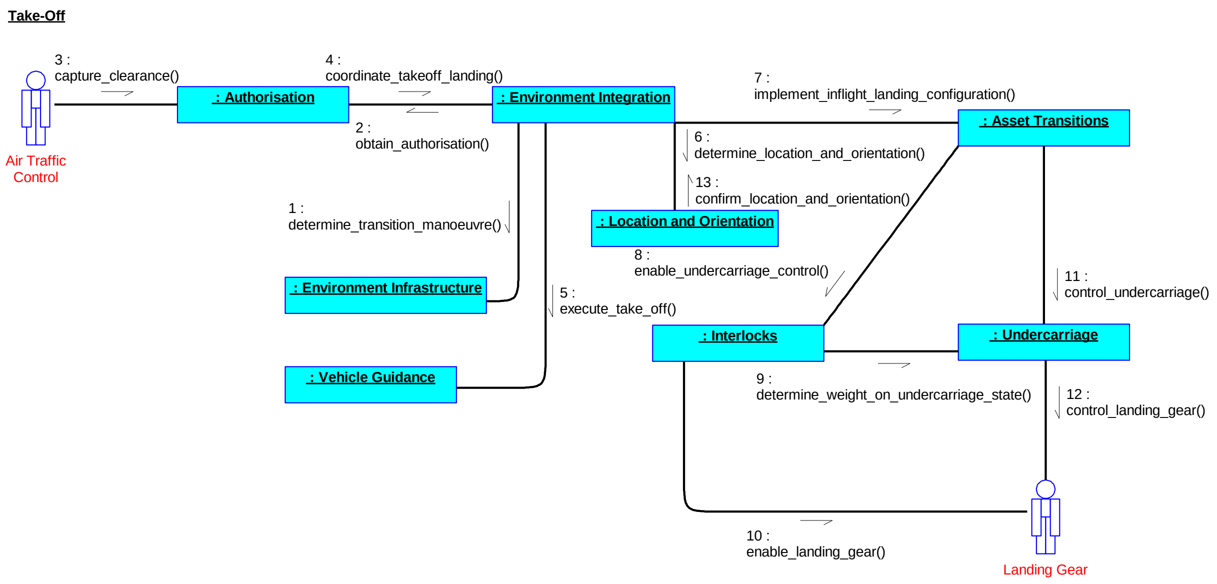

Take-Off
3 : capture_clearance()
4 : coordinate_takeoff_landing()
7 : implement_inflight_landing_configuration()
: Authorisation
: Environment Integration
: Asset Transitions
Air Traffic Control
2 : obtain_authorisation()
6 : determine_location_and_orientation()
13 : confirm_location_and_orientation()
1 : determine_transition_manoeuvre()
: Location and Orientation
8 : enable_undercarriage_control()
11 : control_undercarriage()
: Environment Infrastructure
5 : execute_take_off()
: Interlocks
: Undercarriage
: Vehicle Guidance
9 : determine_weight_on_undercarriage_state()
12 : control_landing_gear()
10 : enable_landing_gear()
Landing Gear

Figure 147: Take-Off IV

Environment Integration begins coordination of the take-off by requesting Environment Infrastructure to determine the required take-off profile, the details of which is passed to Environment Integration as it coordinates the take off. Environment Integration requests authorisation to commence take-off from Authorisation. Upon receiving take off clearance from Air Traffic Control, Environment Integration provides the necessary detail and asks Vehicle Guidance to execute the take-off profile. Vehicle

Guidance, using detailed demand information provided by Environment Infrastructure, generates the take-off manoeuvre and controls the vehicle.

Once the necessary flight criteria are achieved, e.g. appropriate altitude, Environment Integration requests Asset Transitions to transition the vehicle to an in-flight configuration. Asset Transitions requests Interlocks to enable the control of the undercarriage. Undercarriage determines that the weight is off the undercarriage and Interlocks enables control. Asset Transitions requests that Undercarriage retracts the Landing Gear.

Location and Orientation confirms to Environment Integration that cruising altitude has been reached to conclude take off.

C.2.1.2.1.3 Post-Conditions

- The aircraft has taken off and is proceeding with departure.

C.2.1.2.1.4 Actors

Air Traffic Control

The external authority with responsibility for providing take-off and landing clearances.

Landing Gear

The landing gear of the undercarriage.

C.2.1.2.2 Landing

This IV illustrates the process of coordinating and executing the transition of an air vehicle from air to ground. It covers the coordination of the conditions needed for landing, performing the landing manoeuvre and controlling the changes needed to the aircraft configuration prior to landing.

It is assumed that:

- The Exploiting Platform is a conventional-landing aircraft that lands on a runway.
- This is a successful landing of the aircraft, with no need to abort the landing.
- There are no adverse weather conditions.
- The landing is performed using Ground Based Augmentation System landing.

The following related areas of functionality are excluded:

- Any need for an emergency landing.
- The earlier stages of approach (i.e. any manoeuvres prior to initial approach fix, transit to airfield airspace and holding patterns), and any interactions with Air Traffic Control or additional manoeuvres that would be associated with this.
- Handover from central control to local control for the purposes of landing.
- Taxi from the runway.

C.2.1.2.2.1 Pre-Conditions

- The aircraft has completed its main approach and is ready to obtain final approach fix and commence landing.
- The aircraft has received an operator instruction to land.

C.2.1.2.2.2 View

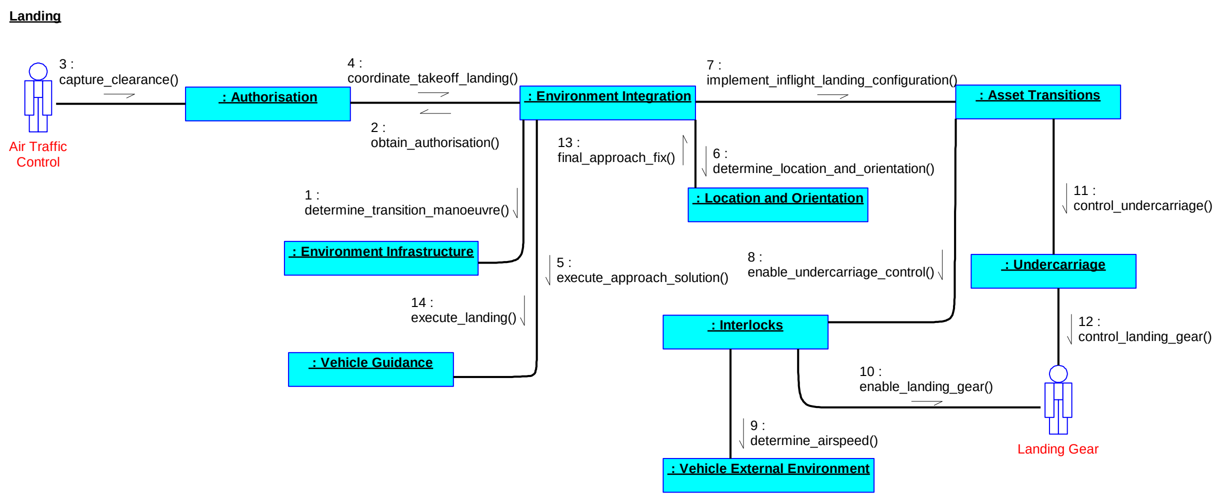

Landing
3 : capture_clearance()
Air Traffic Control
: Authorisation
4 : coordinate_takeoff_landing()
2 : obtain_authorisation()
: Environment Integration
7 : implement_inflight_landing_configuration()
: Asset Transitions
13 : final_approach_fix()
6 : determine_location_and_orientation()
: Location and Orientation
1 : determine_transition_manoeuvre()
: Environment Infrastructure
11 : control_undercarriage()
5 : execute_approach_solution()
8 : enable_undercarriage_control()
: Undercarriage
14 : execute_landing()
: Interlocks
12 : control_landing_gear()
: Vehicle Guidance
10 : enable_landing_gear()
Landing Gear
9 : determine_airspeed()
: Vehicle External Environment

Figure 148: Landing IV

Environment Integration begins coordination of the landing by requesting Environment Infrastructure to determine the required landing profile, the details of which are provided to Environment Integration as it coordinates the landing. Environment Integration requests authorisation to commence landing from Authorisation. Upon receiving landing clearance from Air Traffic Control, Environment Integration provides the necessary detail and asks Vehicle Guidance to execute the landing profile. Vehicle Guidance, using detailed demand information provided by Environment Integration, generates the approach manoeuvres and controls the vehicle.

Once the necessary flight criteria are achieved, e.g. appropriate altitude, Environment Integration requests Asset Transitions to transition the vehicle to a touch-down configuration. Asset Transitions requests Interlocks to enable the control of the undercarriage. Interlocks uses Vehicle External Environment to determine that the aircraft is travelling at an appropriate speed to extend the undercarriage and enables control. Asset Transitions requests Undercarriage to extend the Landing Gear.

Location and Orientation informs Environment Integration that the platform has obtained final approach fix. Environment Integration then instructs Vehicle Guidance to perform the landing manoeuvre, the landing profile completes when the vehicle touches down and reduces speed.

C.2.1.2.2.3 Post-Conditions

- The aircraft has landed safely and is ready to taxi.

C.2.1.2.2.4 Actors

Air Traffic Control

The external authority with responsibility for providing take-off and landing clearances.

Landing Gear

The landing gear of the undercarriage.

C.2.1.3 Routing

The mission requires a task to be performed by a vehicle that requires a route to visit a number of waypoints. Constraints (e.g. air volumes, terrain and speed restrictions) and limits (vehicle capability) apply to the generation of the route.

This scenario assumes a tactically benign environment, i.e. there are no threats present. It also assumes the presence of an independent route checker to provide a high integrity guarantee of no conflict with terrain, obstructions, unauthorised airspace, etc.

The following related areas of functionality are excluded:

- The reason for the route being requested.
- How the route is followed by a vehicle.
- Determination of the fuel cost for traversing the route.

C.2.1.3.1 Pre-Conditions

- A task that requires routing as part of its solution has been received.

C.2.1.3.2 View

Routing

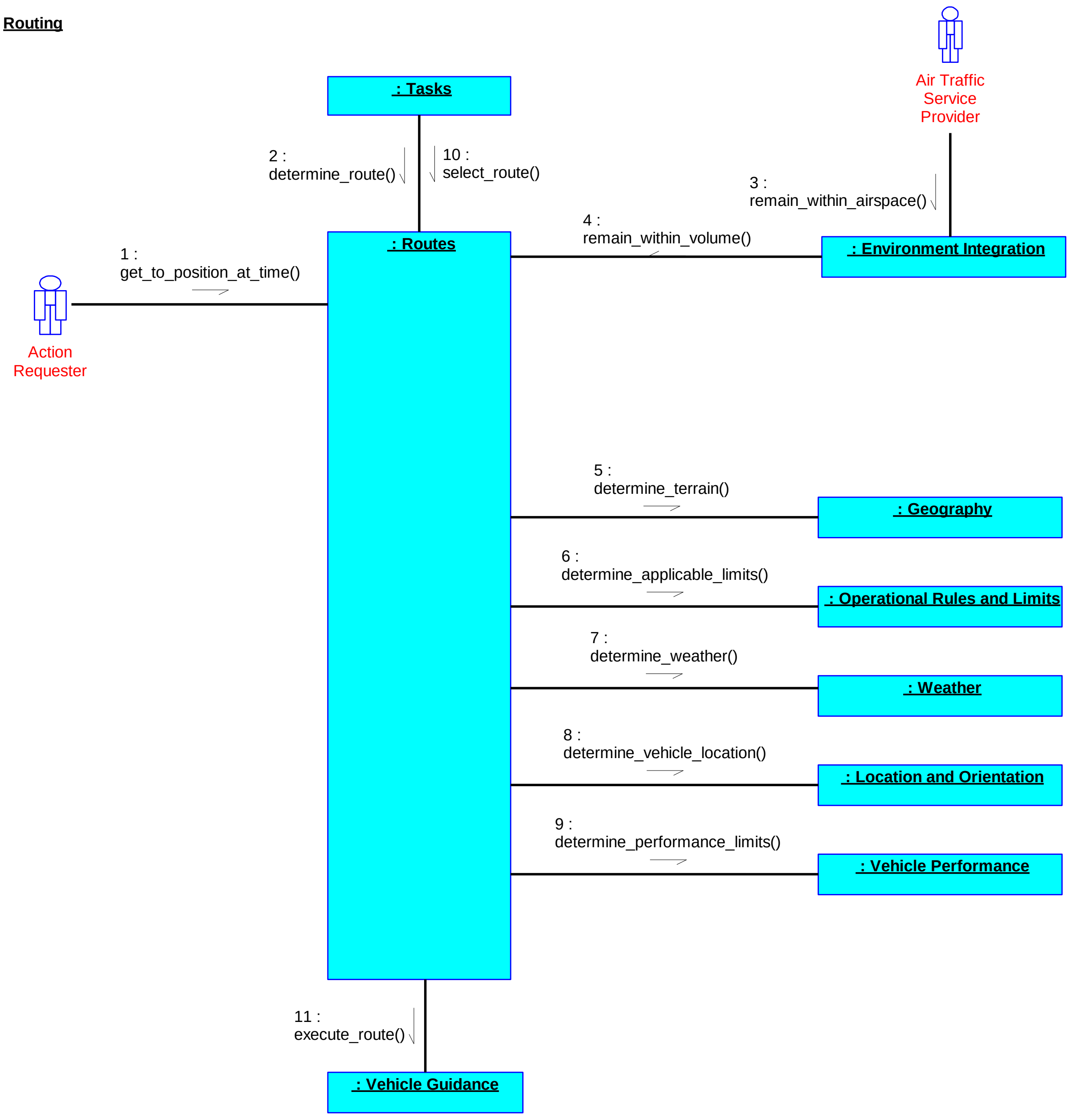

: Tasks
Air Traffic Service Provider
2 : determine_route()
10 : select_route()
3 : remain_within_airspace()
4 : remain_within_volume()
1 : get_to_position_at_time()
: Routes
: Environment Integration
Action Requester
5 : determine_terrain()
: Geography
6 : determine_applicable_limits()
: Operational Rules and Limits
7 : determine_weather()
: Weather
8 : determine_vehicle_location()
: Location and Orientation
9 : determine_performance_limits()
: Vehicle Performance
11 : execute_route()
: Vehicle Guidance

Figure 149: Routing IV

Requirements are placed upon Routes from a number of sources:

- Tasks places requirements upon Routes to achieve a number of positions.
- Action Requester requests Routes to position the vehicle such that an action can be performed at a particular time.

Routes is made aware of a number of factors that will constrain the route including:

- Air volumes to remain within.
- Regions to avoid.
- Rules of engagement stating that supersonic flight is prohibited within a certain region.

- Weather patterns which may need to be avoided.

Tasks asks Routes to determine whether a route can be generated that meets both the requirements within the constraints.

Routes generates a route from the current location (as provided by Location and Orientation) to the required position, considering the vehicle movement capability (as provided by Vehicle Performance).

The achievability of the route is reported to Tasks. Tasks accepts the route for execution and Routes informs Vehicle Guidance to execute the trajectory demand.

C.2.1.3.3 Post-Conditions

- An achievable route that meets the positional requirements and constraints and is within limits has been selected for execution.

C.2.1.3.4 Actors

Action Requester

A component that places requirements upon Routes so that actions can be performed at a particular time.

Air Traffic Service Provider

The ATS with responsibility for providing information and instructions required to comply with airspace rules and restrictions.

C.2.1.4 Vehicle Movement

C.2.1.4.1 Vehicle Movement in Air

This IV describes how a vehicle's movement is controlled whilst in the air (i.e. aviating), based on the receipt of a movement demand.

It is assumed that:

- Air vehicle control is via actuator-driven control surfaces and a propulsion unit.
- All direct control feeds into Vehicle Guidance, rather than any control demands feeding directly to Vehicle Stability and Control.
- There are no system health issues.
- The movement demands generated do not come from an assured source that can be relied upon to avoid terrain collision and avoid no fly zones.

The following related areas of functionality are excluded:

- Transition from flight to taxiing or vice-versa.
- The use of sensing and/or communicators in establishing a movement demand.
- Determination of air vehicle location and orientation, and vehicle external environment parameters.
- Determination of vehicle performance limits.
- Mechanical Positioning control of Effectors.
- Collision prediction (this scenario is focussed on a 'time horizon' that does not require any collision prediction functionality, see the Figure 156: Terrain Avoidance IV for a scenario that includes collision prediction and avoidance functionality).

C.2.1.4.1.1 Pre-Conditions

- Conflict Resolution has been given appropriate arbitration rules.

C.2.1.4.1.2 View

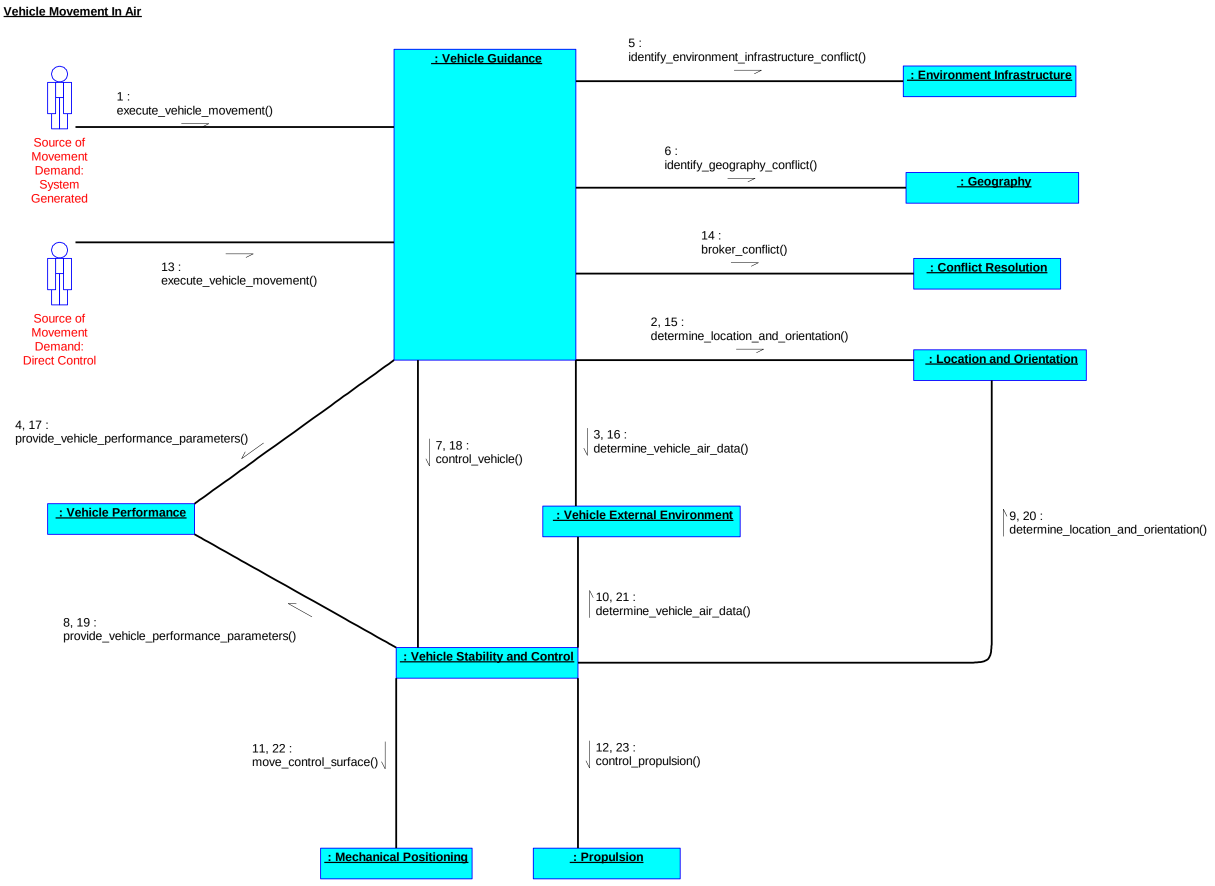

Vehicle Movement In Air
: Vehicle Guidance
Source of Movement Demand: System Generated
1 : execute_vehicle_movement()
Source of Movement Demand: Direct Control
13 : execute_vehicle_movement()
5 : identify_environment_infrastructure_conflict()
: Environment Infrastructure
6 : identify_geography_conflict()
: Geography
14 : broker_conflict()
: Conflict Resolution
2, 15 : determine_location_and_orientation()
: Location and Orientation
4, 17 : provide_vehicle_performance_parameters()
7, 18 : control_vehicle()
3, 16 : determine_vehicle_air_data()
: Vehicle Performance
: Vehicle External Environment
9, 20 : determine_location_and_orientation()
10, 21 : determine_vehicle_air_data()
8, 19 : provide_vehicle_performance_parameters()
: Vehicle Stability and Control
11, 22 : move_control_surface()
12, 23 : control_propulsion()
: Mechanical Positioning
: Propulsion

Figure 150: Vehicle Movement In Air IV

A demand to alter the flight path of the air vehicle is initially provided as a system generated movement demand (from Source of Movement Demand: System Generated).

Vehicle Guidance receives the vehicle movement demand and generates the vehicle location and attitude movement profile, which forms the basis of any vehicle location and attitude demands provided to Vehicle Stability and Control later on. In order to do this, and apply any necessary corrections over time, it requires information about the air vehicle and the performance envelope of the air vehicle, which is obtained as follows:

- Air vehicle location and orientation and air data parameters are determined by Location and Orientation and Vehicle External Environment respectively.
- Vehicle performance envelope (flight envelope) parameters are obtained from Vehicle Performance.

Since the vehicle movement demand, received by Vehicle Guidance, has not undergone the required integrity checks, Vehicle Guidance requests that Geography and Environment Infrastructure identify any conflicts between terrain or no fly zones and the proposed vehicle location movement profile. These components report back that no conflicts are identified.

Vehicle Guidance provides the vehicle location and attitude movement demands to Vehicle Stability and Control. Vehicle Stability and Control obtains the relevant Vehicle Performance (flight envelope) parameters from Vehicle Performance, as well as the current air vehicle orientation and air data parameters, from Location and Orientation and Vehicle External Environment respectively. These are used by Vehicle Stability and Control to generate the air vehicle control surface and propulsion control demands on Mechanical Positioning and Propulsion.

Control surface actuators are controlled by Mechanical Positioning in order to execute the control surface control demands; similarly propulsion units are controlled by Propulsion in order to execute the propulsion demands.

Later, Vehicle Guidance also receives vehicle movement demands from the operator (generated by a Source of Movement Demand: Direct Control). Since there is a fundamental conflict between the different demands being placed on Vehicle Guidance, Vehicle Guidance requests Conflict Resolution to determine which demands should be acted upon. The arbitration rules within Conflict Resolution dictate that the operator should be given priority, and so Vehicle Guidance is instructed to adhere to the operator demands.

The operator vehicle movement demands are acted upon in the same way as above, for the system generated movement demands, with the exception that checking for conflicts with terrain or no fly zones is not performed, since the operator is responsible for avoiding these.

C.2.1.4.1.3 Post-Conditions

- The air vehicle is aviating along the arbitrated path.

C.2.1.4.1.4 Actors

Source of Movement Demand: Direct Control

An authorised operator that can generate a direct control demand via HMI. This demand includes both a request to select or cancel direct path control and the control demand values (which will affect the trajectory and/or speed of the air vehicle).

Source of Movement Demand: System Generated

Part of the system that can generate a movement demand.

Movement demands include:

- Defined routes (planned, contingency, etc.).
- Avoidance manoeuvres (collision, terrain, weather, threats, etc.).
- Manoeuvre profiles (take-off, landing, etc.).
- Autopilot-type manoeuvres (acquire altitude, hold heading, etc.).

C.2.1.4.2 Vehicle Movement on Ground

This IV describes how movement of an air vehicle whilst on the ground (i.e. taxiing) is controlled, based on the receipt of a movement demand.

It is assumed that:

- Air vehicle control is via actuator-driven controls (such as nose-wheel steering) and a propulsion unit (i.e. the vehicle is not being towed).
- The air vehicle is operating in a fixed-wing, wheels-on-ground taxiing environment only.
- The air vehicle is in a "safe-to-transmit" position/state allowing for remote communications.
- The air vehicle is untethered from any physical cabling and restraints and is free to move.
- All direct control feeds into Vehicle Guidance, rather than some direct control demands (e.g. required speed and steering) feeding directly to Vehicle Stability and Control.
- There are no system health issues.
- The movement demands generated do not come from a assured source that can be relied upon to avoid exclusion zones.

The following related areas of functionality are excluded:

- Transition from flight to taxiing or vice-versa.
- The use of sensing and/or communicators in establishing a movement demand.
- Determination of air vehicle location and orientation parameters.
- Determination of vehicle performance limits.
- Mechanical Positioning control of Effectors.
- Collision prediction (this scenario is focussed on a 'time horizon' that does not require any collision prediction functionality, see the Figure 156: Terrain Avoidance IV for a scenario that includes collision prediction / avoidance functionality).

C.2.1.4.2.1 Pre-Conditions

- Conflict Resolution has been given appropriate arbitration rules.

C.2.1.4.2.2 View

Vehicle Movement On Ground

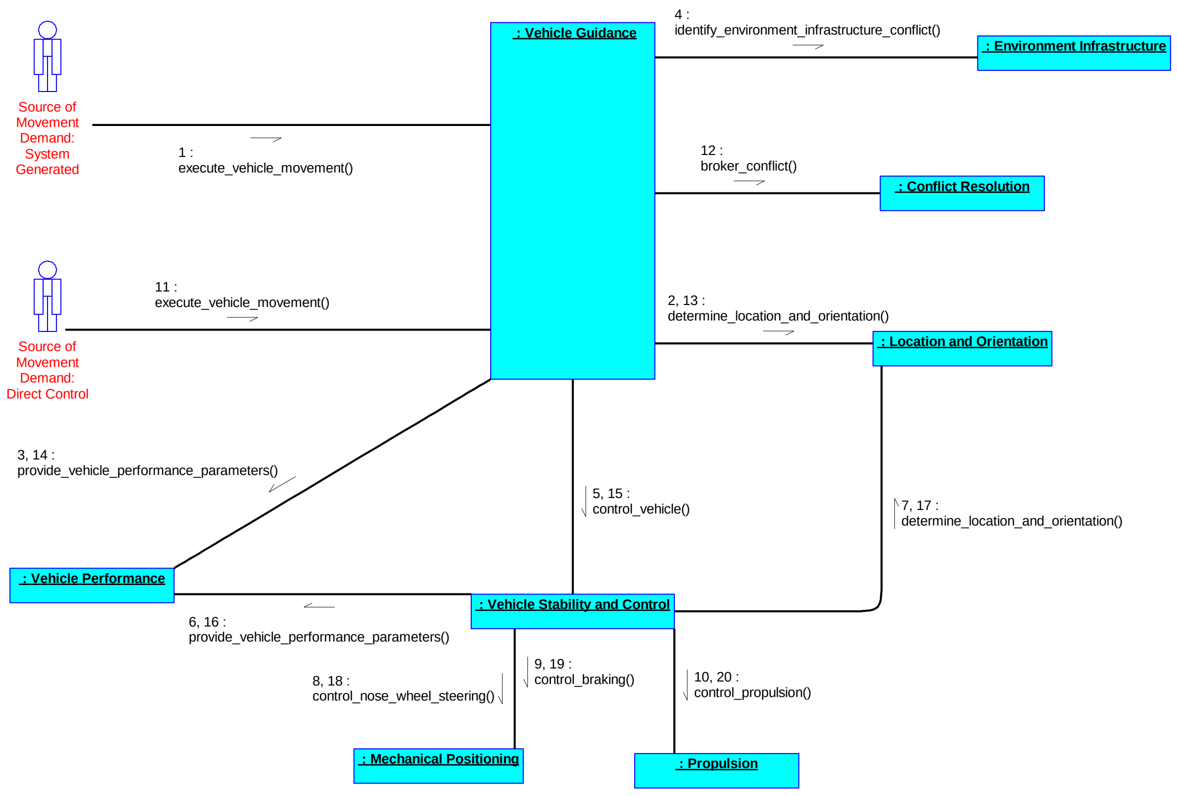

Source of Movement Demand: System Generated
Source of Movement Demand: Direct Control
: Vehicle Guidance
: Environment Infrastructure
: Conflict Resolution
: Location and Orientation
: Vehicle Performance
: Vehicle Stability and Control
: Mechanical Positioning
: Propulsion
1 : execute_vehicle_movement()
11 : execute_vehicle_movement()
4 : identify_environment_infrastructure_conflict()
12 : broker_conflict()
2, 13 : determine_location_and_orientation()
3, 14 : provide_vehicle_performance_parameters()
5, 15 : control_vehicle()
7, 17 : determine_location_and_orientation()
6, 16 : provide_vehicle_performance_parameters()
8, 18 : control_nose_wheel_steering()
9, 19 : control_braking()
10, 20 : control_propulsion()

Figure 151: Vehicle Movement On Ground IV

A demand to alter the taxi path of the vehicle is initially provided as a system generated movement demand (from Source of Movement Demand: System Generated).

Vehicle Guidance receives the vehicle movement demand and generates the vehicle movement profile, which forms the basis of any demands provided to Vehicle Stability and Control later on. In order to do this, and apply any necessary corrections over time, it requires information about the performance envelope of the air vehicle, which is obtained as follows:

- Air vehicle location and orientation is determined by Location and Orientation.
- Vehicle performance envelope parameters are obtained from Vehicle Performance.

Since the vehicle movement demand, received by Vehicle Guidance, has not undergone the required integrity checks, Vehicle Guidance requests that Environment Infrastructure identify any conflicts with exclusion zones and the proposed vehicle location movement profile. The component reports back that no conflicts are identified.

Vehicle Guidance provides the vehicle movement demands to Vehicle Stability and Control. Vehicle Stability and Control obtains the relevant vehicle performance envelope parameters from Vehicle Performance, as well as the current air vehicle orientation from Location and Orientation. These are used by Vehicle Stability and Control to generate the air vehicle braking and nose wheel control demands on Mechanical Positioning as well as propulsion demands on Propulsion.

Mechanical Positioning will control taxiing-specific actuators (e.g. nose wheel steering and brakes) and propulsion units are controlled by Propulsion in order to execute the propulsion demands.

Later, Vehicle Guidance also receives vehicle movement demands from the operator (generated by a Source of Movement Demand: Direct Control). Since there is a fundamental conflict between the different demands being placed on Vehicle Guidance, Vehicle Guidance requests Conflict Resolution to determine which demands should be acted upon. The arbitration rules within Conflict Resolution dictate that the operator should be given priority, and so Vehicle Guidance is instructed to adhere to the operator demands.

The operator vehicle movement demands are acted upon in the same way as above, for the system generated movement demands, with the exception that checking for conflicts with Environment Infrastructure, since the operator is responsible for avoiding this.

C.2.1.4.2.3 Post-Conditions

- The air vehicle is taxiing along the arbitrated path.

C.2.1.4.2.4 Actors

Source of Movement Demand: Direct Control

An authorised operator that can generate a direct control demand via HMI. This demand includes both a request to select or cancel direct path control and the control demand values (which will affect the trajectory and/or speed of the air vehicle).

Source of Movement Demand: System Generated

Part of the system that can generate a movement demand.

Movement demands include:

- Defined routes (planned, contingency, etc.).
- Avoidance manoeuvres (collision, terrain, weather, threats, etc.).
- Manoeuvre profiles (take-off, landing, etc.).
- Autopilot-type manoeuvres (acquire altitude, hold heading, etc.).

C.2.1.5 Vehicle Performance

This use case describes how the current vehicle performance limits are determined so that Vehicle Guidance can meet trajectory demands placed on it.

It is assumed that:

- The allowable performance envelope is a function of altitude, external temperature, vehicle mass, position of the undercarriage, the external role fit configuration and the need to carry out safe store release. In reality, other inputs may also be required, but this subset is used to describe the possible interactions.

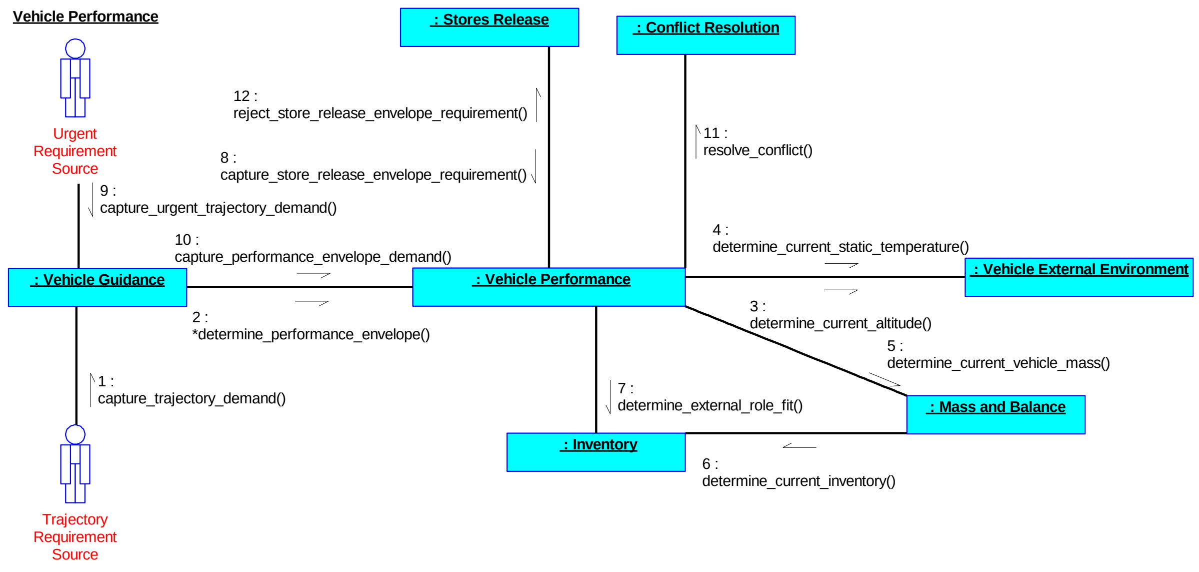
The following related areas of functionality are excluded:

- Consumption of source data for the determination of Vehicle External Environment parameters.
- Controlling the trajectory of the air vehicle.
- Determining the current inventory of the air vehicle.
- Determining a vehicle performance parameter at some point in the future - i.e. when the current conditions do not apply. In this case it is expected that a component requesting this information would define the conditions applicable to the determination of the particular performance parameter (e.g. undercarriage position, vehicle mass or temperature).

C.2.1.5.1 Pre-Conditions

- The air vehicle is airborne.
- A required vehicle trajectory has been determined by the Trajectory Requirement Source.

C.2.1.5.2 View



Vehicle Performance
Urgent Requirement Source
: Stores Release
: Conflict Resolution
12 : reject_store_release_envelope_requirement()
8 : capture_store_release_envelope_requirement()
11 : resolve_conflict()
9 : capture_urgent_trajectory_demand()
10 : capture_performance_envelope_demand()
4 : determine_current_static_temperature()
: Vehicle Guidance
: Vehicle Performance
: Vehicle External Environment
2 : *determine_performance_envelope()
3 : determine_current_altitude()
5 : determine_current_vehicle_mass()
1 : capture_trajectory_demand()
7 : determine_external_role_fit()
: Mass and Balance
: Inventory
6 : determine_current_inventory()
Trajectory Requirement Source

Figure 152: Vehicle Performance IV

A trajectory requirement is received from the Trajectory Requirement Source by Vehicle Guidance. Vehicle Guidance requests the relevant performance envelope from Vehicle Performance. Vehicle

Performance determines the performance envelope for the current conditions such as the current altitude and static temperature (determined by Vehicle External Environment), the current vehicle mass (determined by Mass and Balance based on the current inventory (determined by Inventory)) and the allowable carriage envelope for the current external role fit items (determined by Inventory).

Subsequently, Stores Release determines that there is a need to release a vehicle store. This release activity requires the vehicle to remain within a safe release envelope for that store. The requirement for this activity is captured by Vehicle Performance, which results in a change to the performance envelope useable by Vehicle Guidance until the release activity is completed.

An urgent trajectory demand is then received by Vehicle Guidance (for example, a collision avoidance manoeuvre). Vehicle Guidance determines that it cannot satisfy this requirement within the current performance envelope provided by Vehicle Performance and places a demand on Vehicle Performance to increase the performance envelope.

Vehicle Performance determines that the stores release activity is limiting the performance envelope below the requested value and tasks Conflict Resolution to resolve the conflict. Conflict Resolution resolves the conflict, arbitrating in favour of the urgent trajectory requirement. As a result Vehicle Performance rejects the requirement for a store release envelope (which will delay or cancelling the stores release activity) and is able to increase the performance envelope in line with the demand.

The new performance envelope allows Vehicle Guidance to satisfy the urgent demand placed on it.

C.2.1.5.3 Post-Conditions

- The Vehicle Guidance component knows the performance envelope for the current conditions and so can maintain the air vehicle within the vehicle performance limits.
- The current performance envelope determined by Vehicle Performance allows Vehicle Guidance to satisfy the urgent requirement placed on it.

C.2.1.5.4 Actors

Trajectory Requirement Source

The source of a trajectory requirement.

C.2.2 Vehicle Environment Views

This section contains IVs relating to the environment in which an air vehicle is operating.

The Figure 153: Vehicle Environment Views UC Diagram shows all the use cases within the Vehicle Environment Views and any actors that interact with them.

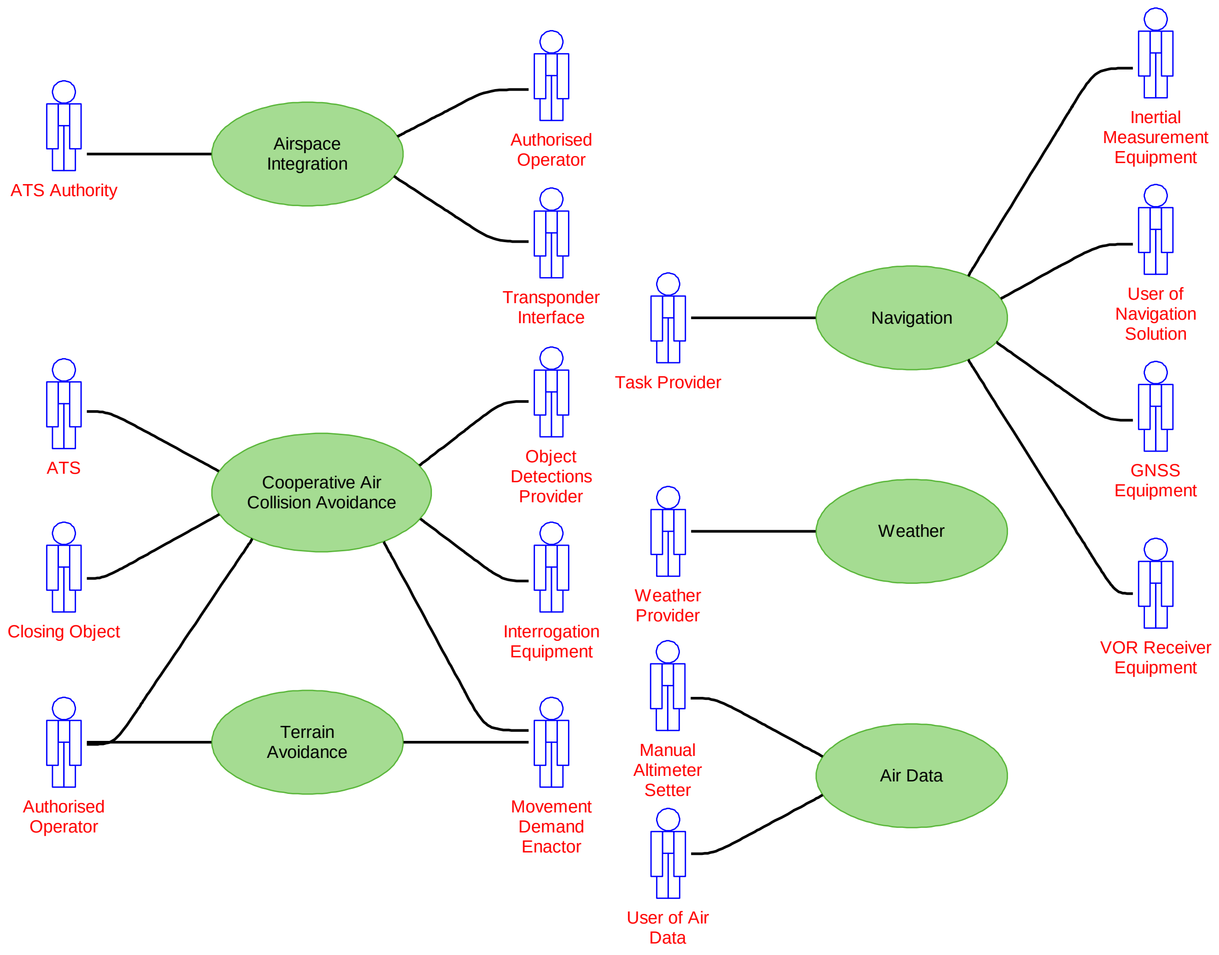


Figure 153: Vehicle Environment Views UC Diagram

C.2.2.1 Airspace Integration

This IV illustrates activities for integrating into controlled airspace:

- Communicating with ATS to request and obtain clearances, in order to comply with airspace restrictions, and to receive instructions upon approaching a controlled airspace boundary.
- Implementing changes to communications and identification when transitioning between ATS controllers.

This IV assumes:

- ATS will initiate the transfer from one air traffic control unit to another when the vehicle approaches the boundary between control areas.

The following considerations are excluded:

- Detection of potential collisions and subsequent determination of an avoidance manoeuvre.

- Determining and following a route.
- Communications between the authorised operator and ATS.
- Authorised operator interactions with the HMI.
- Set-up of requirements to facilitate a new communications channel and determining that the new channel is permitted.
- Initial processing and semantic translation of ATS instructions.

C.2.2.1.1 Pre-Conditions

- The system is pre-loaded with transponder codes and ATS communications frequencies.

C.2.2.1.2 View

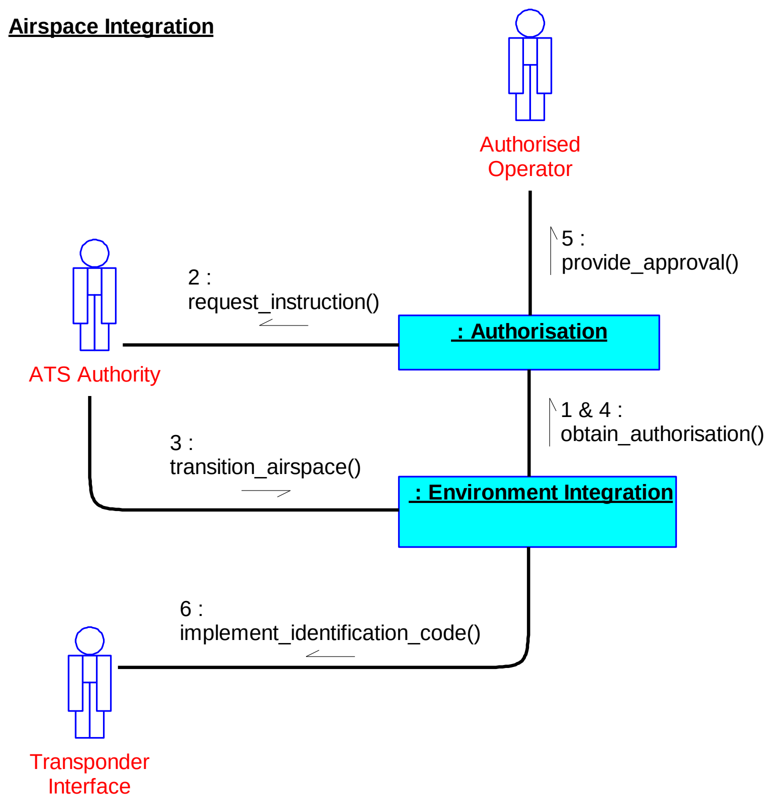

Airspace Integration
Authorised Operator
5 : provide_approval()
2 : request_instruction()
ATS Authority
: Authorisation
1 & 4 : obtain_authorisation()
3 : transition_airspace()
: Environment Integration
6 : implement_identification_code()
Transponder Interface

Figure 154: Airspace Integration IV

Whilst transiting through controlled airspace, Environment Integration requests authorisation for required deviations from the cleared route (e.g. due to weather), which Authorisation obtains by requesting clearance from ATS.

Upon approaching the boundary between control areas, ATS initiates a handover of the air vehicle from the releasing air traffic control unit to a receiving air traffic control unit. Environment Integration interprets the handover instructions from ATS to determine the system setup (e.g. new radio channel or IFF codes) required to comply with them. Environment Integration requests authorisation to enact the communications and identification changes received from ATS for the transition, which Authorisation obtains by requesting approval from an Authorised Operator with an appropriate role.

Environment Integration requests the Transponder Interface to implement the new identification code received from ATS.

C.2.2.1.3 Post-Conditions

- The system has transitioned from one area of controlled airspace to another.

C.2.2.1.4 Actors

ATS Authority

The external authority with responsibility for providing information and instructions required to comply with airspace rules and restrictions.

Authorised Operator

An authorised operator within a system with the authority to approve ATS instructions.

Transponder Interface

An interface to an external transponder that enables interactions with other airspace users.

C.2.2.2 Avoidance

C.2.2.2.1 Aircraft Collision Avoidance

This IV describes co-operative collision avoidance between ownship and another air vehicle. Collision Prediction uses vehicle trajectories to identify potential conflicts. In addition, Environment Integration coordinates the provision of information to other airspace users about the location, heading and speed of ownship. When a potential conflict is identified, Collision Avoidance determines an appropriate manoeuvre to avoid the collision.

In the case of Airborne Collision Avoidance System (ACAS), following a Resolution Advisory (RA), Collision Avoidance generates an avoidance manoeuvre, coordinating with the intruder according to published ACAS logic. Where the intruder has a transponder but no ACAS, only ownship will manoeuvre.

This use case describes how the flight path of the vehicle is adapted to cater for the presence of an intruder in the protected volume. The use case described in this IV is for an airborne example. Similar principles would apply for ground based avoidance.

It is assumed that:

- Collision Prediction and Collision Avoidance are compliant with the ACAS requirement (minimum ACAS II), and avoidance is achieved entirely through the use of ACAS.
- The avoiding action is taken autonomously.
- Received transponder information will not need tactical processing for collision avoidance purposes.

The following considerations are excluded from this Use Case:

- Tactical interactions.
- Sensing interactions.
- HMI interactions.
- Use of the information to create or update tracks for purposes other than collision avoidance.
- Human control of the air vehicle and human interactions with Air Traffic Services (ATS) or proximate traffic, etc.
- Recovery to the required route and clearance following the avoidance manoeuvre.
- The use of the Trajectory Prediction component since avoidance is achieved, in this scenario, through the use of ACAS.

C.2.2.2.1.1 Pre-Conditions

- The air vehicle is airborne and allocated to an air traffic controller.
- The EMCON restrictions under which the air vehicle is operating permit the appropriate hardware to transmit.

C.2.2.2.1.2 View

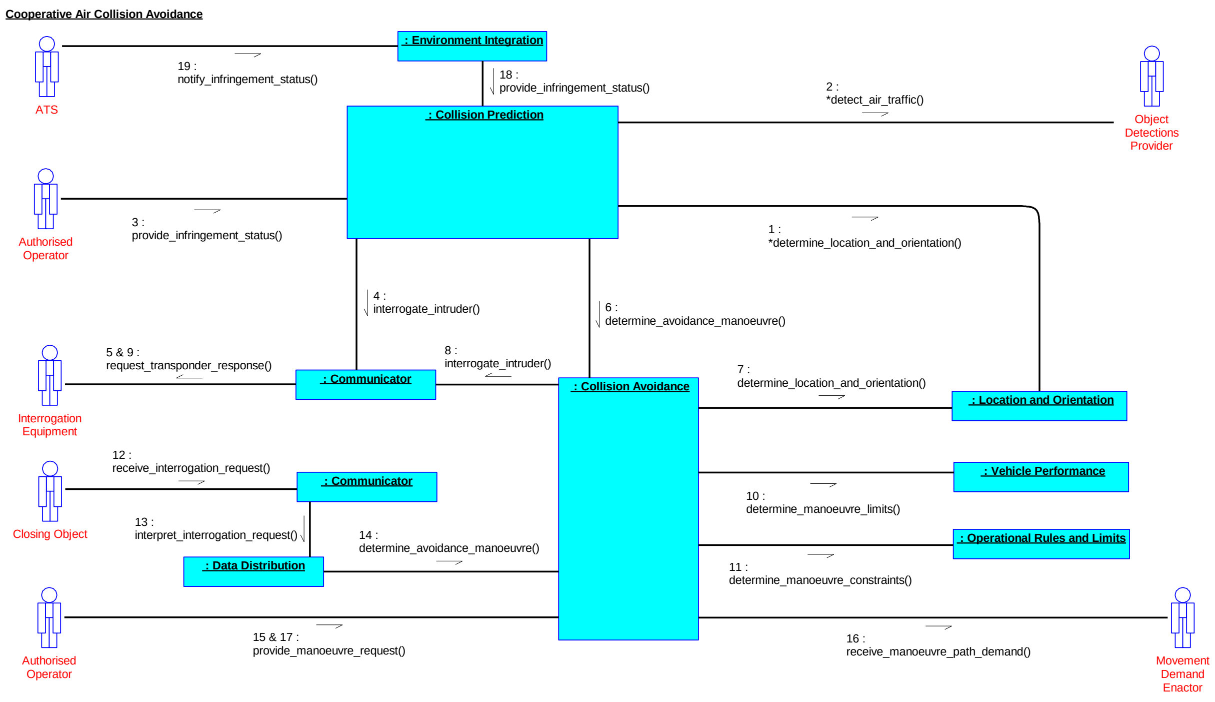

Cooperative Air Collision Avoidance
ATS
Authorised Operator
Interrogation Equipment
Closing Object
Authorised Operator
Object Detections Provider
Movement Demand Enactor
: Environment Integration
: Collision Prediction
: Communicator
: Communicator
: Data Distribution
: Collision Avoidance
: Location and Orientation
: Vehicle Performance
: Operational Rules and Limits
1 : *determine_location_and_orientation()
2 : *detect_air_traffic()
3 : provide_infringement_status()
4 : interrogate_intruder()
5 & 9 : request_transponder_response()
6 : determine_avoidance_manoeuvre()
7 : determine_location_and_orientation()
8 : interrogate_intruder()
10 : determine_manoeuvre_limits()
11 : determine_manoeuvre_constraints()
12 : receive_interrogation_request()
13 : interpret_interrogation_request()
14 : determine_avoidance_manoeuvre()
15 & 17 : provide_manoeuvre_request()
16 : receive_manoeuvre_path_demand()
18 : provide_infringement_status()
19 : notify_infringement_status()

Figure 155: Cooperative Air Collision Avoidance IV

Collision Prediction constantly monitors ownship position and predicted movement in relation to other objects in the environment. Collision Prediction acquires ownship position and velocity vector from Location and Orientation. Collision Prediction receives notification of other objects in the environment from the Object Detections Provider. As a result, Collision Prediction determines that a Closing Object will infringe the Traffic Advisory (TA) region of the protected volume, thus becoming an intruder. Collision Prediction therefore alerts the Authorised Operator to the presence of the Closing Object via a TA.

When the intruder enters the RA region, Collision Avoidance interrogates the intruder and generates an RA alert to coordinate an avoidance manoeuvre with the intruder. This manoeuvre is based on ownship information from Location and Orientation, with the manoeuvre (climb or descend) being provided in the interrogation via Communicator (based on the available position information of both the aircraft, including height above ground, speed and heading). Applicable constraints from Vehicle Performance and Operational Rules and Limits (e.g. to avoid breaching limits) may influence the ownship manoeuvre.

If the intruder initiates the manoeuvre first, this information is received by Communicator via an interrogation request, then interpreted by Data Distribution and passed to Collision Avoidance to trigger the coordinated manoeuvre.

The Authorised Operator is alerted to the proximity of the aircraft via an RA, from Collision Avoidance.

Collision Avoidance informs Movement Demand Enactor of the manoeuvre requirement. The manoeuvre is performed, resulting in collision avoidance.

Collision Avoidance notifies the Authorised Operator that the conflict has cleared and an autonomous avoidance manoeuvre has been conducted.

Data from Collision Prediction is recorded to support the reporting of an Airprox incident to ATS; where appropriate, Environment Integration notifies the ATS of the incident and if the prescribed flight level has been contravened by the avoidance manoeuvre.

C.2.2.2.1.3 Post-Conditions

- Collision avoided, air vehicle continues to be airborne and monitored by an air traffic controller.

C.2.2.2.1.4 Actors

ATS

The ATS controlling the airspace.

Authorised Operator

The authorised operator (e.g. the pilot) of the ownship, located within the air vehicle or otherwise.

Closing Object

The object closing with the protected volume and therefore posing the collision risk. The closing object will become the intruder whilst within the protected volume.

Object Detections Provider

Source of object detections, e.g. Sensors or Tactical Objects.

Movement Demand Enactor

The enactor for movement demands that can be generated by the system or by an Authorised Operator via HMI.

Movements demands include:

- Defined routes (planned, contingency, etc.).
- Avoidance manoeuvres (collision, terrain, weather, threats, etc.).
- Manoeuvre profiles (take-off, landing, etc.).
- Autopilot-type manoeuvres (acquire altitude, hold heading, etc.).

Interrogation Equipment

An IFF interrogator.

C.2.2.2.2 Terrain Avoidance

This IV describes a specific example of potential infringement of the MSD between an air vehicle and the ground being identified by Collision Prediction, based on the current vehicle location and dynamics. Once identified, Collision Avoidance determines an appropriate manoeuvre to avoid the collision.

It is assumed that:

- Terrain avoidance is performed autonomously.
- The route or intended path of the air vehicle, as planned, remains outside of the MSD.
- The air vehicle is not performing terrain following.

The following considerations are excluded:

- HMI interactions.
- Terrain avoidance used to detect divergence from landing aids glideslope.
- Recovery to the required route or clearance following the avoidance manoeuvre.

C.2.2.2.2.1 Pre-Conditions

- The air vehicle is airborne and flying within the operating envelope.
- The air vehicle is not preparing to land.
- The air vehicle has deviated from the intended path.

C.2.2.2.2.2 View

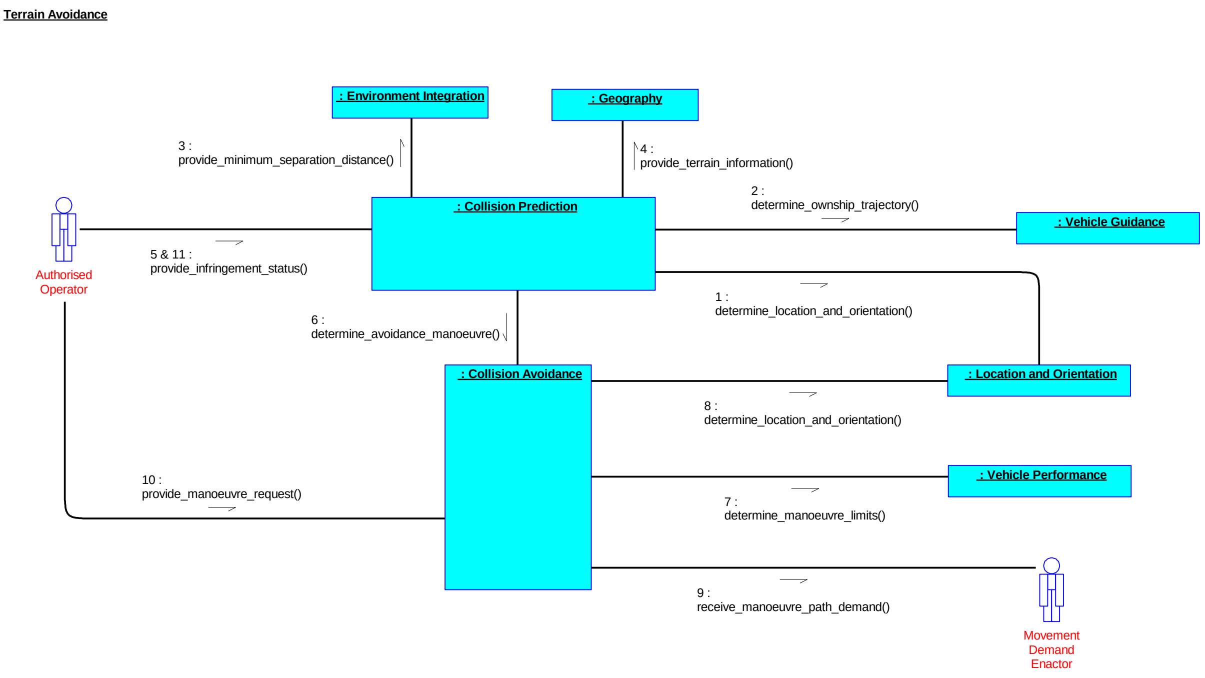

Terrain Avoidance
: Environment Integration
: Geography
3 : provide_minimum_separation_distance()
4 : provide_terrain_information()
: Collision Prediction
2 : determine_ownship_trajectory()
: Vehicle Guidance
Authorised Operator
5 & 11 : provide_infringement_status()
1 : determine_location_and_orientation()
6 : determine_avoidance_manoeuvre()
: Collision Avoidance
: Location and Orientation
8 : determine_location_and_orientation()
10 : provide_manoeuvre_request()
: Vehicle Performance
7 : determine_manoeuvre_limits()
9 : receive_manoeuvre_path_demand()
Movement Demand Enactor

Figure 156: Terrain Avoidance IV

Using ownship information received from Location and Orientation and Vehicle Guidance, Collision Prediction determines that the air vehicle will infringe the MSD between the air vehicle and ground terrain. The MSD between the air vehicle and ground terrain is provided by Environment Integration; with the terrain information provided by Geography.

Collision Prediction alerts the Authorised Operator of the MSD infringement.

Collision Prediction notifies Collision Avoidance of the need to perform a collision avoidance manoeuvre. Collision Avoidance determines the appropriate manoeuvre, applying manoeuvre limits from Vehicle Performance and taking into account ownship information received from Location and Orientation. Collision Avoidance informs Movement Demand Enactor of the manoeuvre requirement, which enacts the avoidance manoeuvre. Collision Avoidance notifies the Authorised Operator that a manoeuvre has been requested. The manoeuvre is performed, the Authorised Operator is informed and safe separation is recovered.

When Collision Prediction no longer predicts an infringement of the MSD, the avoidance demand on Collision Avoidance will be cancelled and the vehicle path demand removed. Collision Prediction will notify the Authorised Operator that the collision risk is cleared.

C.2.2.2.2.3 Post-Conditions

- Infringement of the MSD (terrain or obstacle) has been avoided and the air vehicle continues to fly within the operating envelope.

C.2.2.2.2.4 Actors

Authorised Operator

The authorised operator (e.g. the pilot) of the ownship, located within the air vehicle or otherwise.

Movement Demand Enactor

The enactor for movement demands that can be generated by the system or by an Authorised Operator via HMI.

Movements demands include:

- Defined routes (planned, contingency, etc.).
- Avoidance manoeuvres (collision, terrain, weather, threats, etc.).
- Manoeuvre profiles (take-off, landing, etc.).
- Autopilot-type manoeuvres (acquire altitude, hold heading, etc.).

C.2.2.3 Navigation

This IV illustrates an example of the use of navigation sensors to establish a navigation solution that provides the location, orientation and rates of change of the air vehicle. The current navigation solution accuracy does not meet the new situational needs so a new demand to determine a navigation solution to a specified accuracy is made.

It is assumed that:

- Human interaction to accept or manually update position fixing is not required.
- EMCON does not preclude the use of any available sensor.
- The specific location and orientation parameters and references, and the methods used to derive them lie within the Location and Orientation component itself.
- The location of all possible geo-located navigation aids (e.g. beacons) is known to the system.
- Maps relating magnetic variation or true north to locations on an Earth model are loaded within the system.

The following considerations are excluded:

- Inputs due to sensor detection processing (e.g. terrain map matching) - the scenario uses inputs provided by sensors only.
- Resource contention and resource allocation.
- The impact of the navigation solution using sensors that may be needed for other tasks including reporting the cost of using these sensors.

C.2.2.3.1 Pre-Conditions

- There is an existing navigation solution that provides the current air vehicle location.

C.2.2.3.2 View

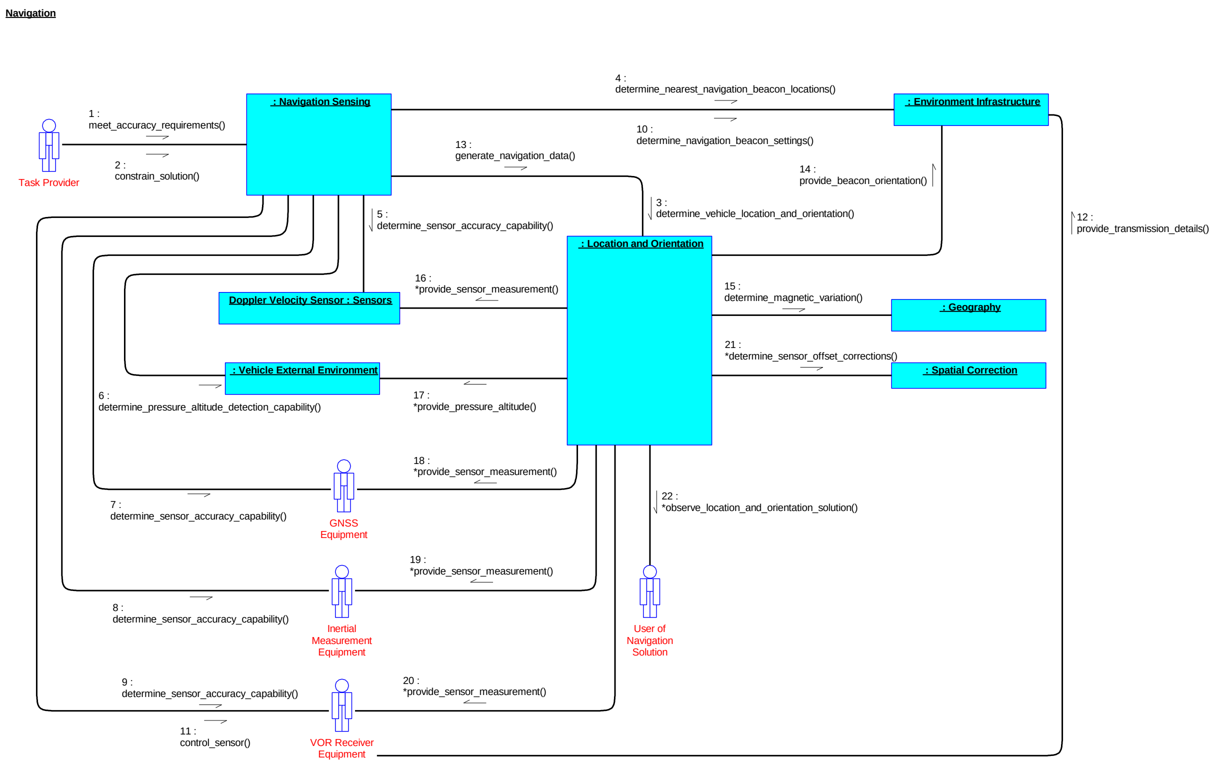

Navigation
: Navigation Sensing
: Environment Infrastructure
: Location and Orientation
Doppler Velocity Sensor : Sensors
: Vehicle External Environment
: Geography
: Spatial Correction
Task Provider
GNSS Equipment
Inertial Measurement Equipment
VOR Receiver Equipment
User of Navigation Solution
1 : meet_accuracy_requirements()
2 : constrain_solution()
3 : determine_vehicle_location_and_orientation()
4 : determine_nearest_navigation_beacon_locations()
5 : determine_sensor_accuracy_capability()
6 : determine_pressure_altitude_detection_capability()
7 : determine_sensor_accuracy_capability()
8 : determine_sensor_accuracy_capability()
9 : determine_sensor_accuracy_capability()
10 : determine_navigation_beacon_settings()
11 : control_sensor()
12 : provide_transmission_details()
13 : generate_navigation_data()
14 : provide_beacon_orientation()
15 : determine_magnetic_variation()
16 : *provide_sensor_measurement()
17 : *provide_pressure_altitude()
18 : *provide_sensor_measurement()
19 : *provide_sensor_measurement()
20 : *provide_sensor_measurement()
21 : *determine_sensor_offset_corrections()
22 : *observe_location_and_orientation_solution()

Figure 157: Navigation IV

Navigation Sensing receives a demand from the Task Provider to establish a navigation solution that provides the location, orientation and rates of change of the air vehicle to a specified accuracy. Navigation Sensing also receives a constraint from the Task Provider to not require any specific air vehicle positioning or manoeuvres in support of generating a navigation solution.

To aid in determining the options available at the current location, Navigation Sensing requests the current air vehicle position from Location and Orientation, which Location and Orientation then provides based on the existing navigation solution.

Navigation Sensing requests Environment Infrastructure to provide information about any TACAN or VOR beacons within a specified region (centred on the current air vehicle location). Environment Infrastructure identifies two VOR beacons matching the criteria and provides the requested information including their coverage.

Navigation Sensing requests the potential navigation sensor sources to provide a statement of their capability.

Navigation Sensing determines a combination of available sensor capabilities which when combined has the capability to meet the required navigation solution accuracy and constraints. In this scenario, this includes the following: Doppler Velocity Sensors, GNSS, Inertial Sensor, pressure based altitude and VOR receiver.

The previous requirements placed on the Doppler Velocity Sensor, GNSS, Inertial Sensor and Vehicle External Environment (for pressure altitude) are determined by Navigation Sensing as still being valid to achieve their contribution to the required navigation solution accuracy and constraints. Note that there may be several requirements that can be satisfied simultaneously by a single navigation solution. Navigation Sensing will balance the needs of the different requirements within its solution.

Navigation Sensing requests Environment Infrastructure to determine the properties of the VOR beacons identified as available navigational aids (e.g. frequency and channel). Once these have been determined, Navigation Sensing places requirements on the air vehicle VOR Receiver Equipment to obtain the necessary transmission details from Environment Infrastructure, and use the settings to configure itself to detect these beacons and provide the air vehicle position relative to the VOR beacons.

Once all of the sensor settings have been established, Navigation Sensing provides requirements to Location and Orientation to produce the navigation solution information. This includes a specification of which information sources should be used and if they should be incorporated into the solution or used as checks to ensure that the solution remains within the expected bounds.

Information about the reference orientation of the beacons is provided to Location and Orientation (i.e. whether the beacon is aligned to true or magnetic North). Location and Orientation requests Geography to provide the magnetic variation at the beacon locations to allow the VOR measurements to be adjusted if necessary.

Location and Orientation continuously obtains the information from each source that is needed to produce the navigation solution.

Spatial Correction is used to determine corrections to offsets between the sensors producing measurement information where they are affected by aero-elastic deformation (e.g. when mounted on the wings).

Location and Orientation continuously refines the air vehicle navigation solution using the relevant data from different sources.

Location and Orientation continuously makes available the air vehicle navigation solution information to any User of Navigation Solution that requires it.

In response to its requirement to produce the navigation solution, Location and Orientation continuously reports the achieved accuracy of the solution to Navigation Sensing.

Navigation Sensing reports the status of achieving the required navigation solution to the Task Provider. Navigation Sensing continuously monitors the performance achieved in meeting the requirement and identifies if any further changes to the utilised sensors, their settings and how these should be integrated into a navigation solution are required to maintain the achievement of the requirement over time.

C.2.2.3.3 Post-Conditions

- Navigation sensors have been allocated and settings established.
- Air vehicle navigation solution(s) are providing the required location, orientation and rates of change information to the required accuracy.

C.2.2.3.4 Actors

GNSS Equipment

The interface to the equipment hardware for the GNSS receiver.

Inertial Measurement Equipment

The interface to the equipment hardware for the inertial measurement system.

Task Provider

The source of a demand to generate a navigation solution that provides the location, orientation and rates of change of the air vehicle to a specific accuracy.

User of Navigation Solution

A user of the navigation solution location, orientation and rates of change information (which may be either a person observing the output using a HMI, or another component within the architecture).

VOR Receiver Equipment

The interface to the equipment hardware for the VOR receiver.

C.2.2.4 Weather

The purpose of this IV is to show how the system obtains weather data through sensing and reports and then determines which weather is threatening. It also shows how the system determines the steps to take to mitigate that weather threat.

It is assumed that:

- The Weather Provider updates the system throughout the duration of the mission.

The following related areas of functionality are excluded:

- The communication between external actors and the system.
- Pre-mission planning and showing the loading of mission related data (including weather).
- The components and external equipment that provides the Sensor Products component with data.

C.2.2.4.1 Pre-Conditions

Tasks has placed a requirement on Susceptibility to assess the vehicle's susceptibility to particular weather elements.

C.2.2.4.2 View

Weather

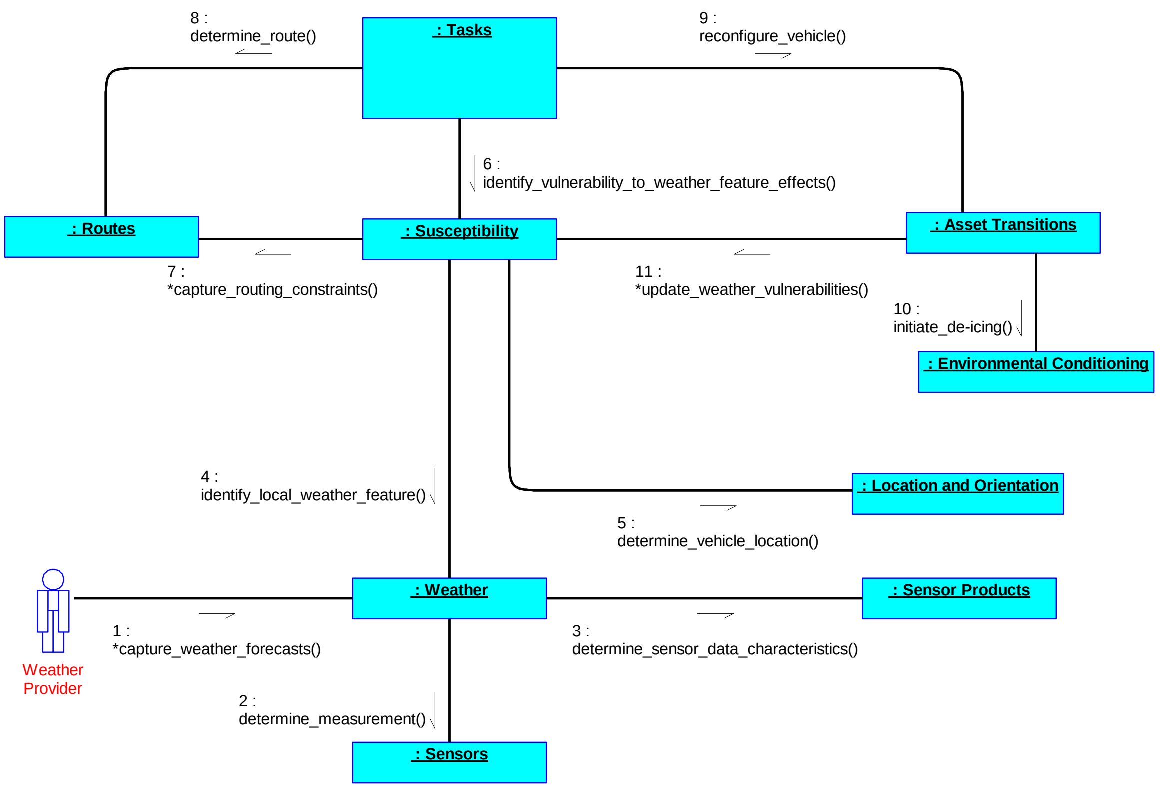

: Tasks
8 : determine_route()
9 : reconfigure_vehicle()
6 : identify_vulnerability_to_weather_feature_effects()
: Routes
: Susceptibility
: Asset Transitions
7 : *capture_routing_constraints()
11 : *update_weather_vulnerabilities()
10 : initiate_de-icing()
: Environmental Conditioning
4 : identify_local_weather_feature()
: Location and Orientation
5 : determine_vehicle_location()
: Weather
: Sensor Products
Weather Provider
1 : *capture_weather_forecasts()
3 : determine_sensor_data_characteristics()
2 : determine_measurement()
: Sensors

Figure 158: Weather IV

Weather obtains data from various different sources; the Weather Provider provides weather reports throughout the mission, Sensors gathers real-time temperature data around the vehicle, Sensor Products collates more complex sensor data, such as cloud formations, which is provided to Weather. Once a weather feature has been identified, it is passed to Susceptibility which assesses the vehicle's susceptibility to particular weather elements. This assessment will be based on knowledge of ownship vulnerabilities in its current state (e.g. icing) to the particular weather element.

As part of this determination of susceptibility the component, with the aid of Location and Orientation, will determine a number of regions in which the vehicle is susceptible and the level of susceptibility it has in these regions. The susceptibility risks are passed up to Tasks which plans or executes actions to reduce the susceptibility of the vehicle. In this case constraints can be given to Routes to avoid a particular region (with this region being updated continuously by Susceptibility). Tasks also places a requirement on Asset Transitions to reconfigure the vehicle to get it into a state which makes it less susceptible; this is facilitated by the use of Environmental Conditioning (e.g. start de-icing). Asset Transitions informs Susceptibility of the change in asset state such that the route can be reassessed based on new vulnerability advice from Susceptibility.

C.2.2.4.3 Post-Conditions

- A weather threat has been identified and the steps to reduce the risk of the threat have been enacted by the system.

C.2.2.4.4 Actors

Weather Provider

A provider of both pre-flight and in-flight information from various sources, including forecast and live weather information.

C.2.2.5 Air Data

This use case describes how air data is determined. Air data includes:

- Calibrated airspeed.
- Mach number.
- Outside air temperature.
- True airspeed.
- Angle of attack (alpha).
- Sideslip (beta).
- Pressure altitude (1013.25mb reference).
- Barometric reference altitude (to a specific pressure reference 1013.25mb, QNH, QFE).

It is assumed that:

- Monitoring of the air external to the air vehicle is accomplished using pressure, temperature and flow angle sensors only.

The following considerations are excluded:

- Determination of the fluid state using laser sensing systems.
- Determination of angle of attack or sideslip using inertial sensors.
- Pressure error corrections reliant on inputs other than pressure, temperature and flow angle.
- Inhibiting and modifying use of particular sensor inputs during firing of weapons and guns where these would adversely affect the sensor.
- Sensor and probe heating.

C.2.2.5.1 Pre-Conditions

- The air system is powered-up and sensors are available.

C.2.2.5.2 View

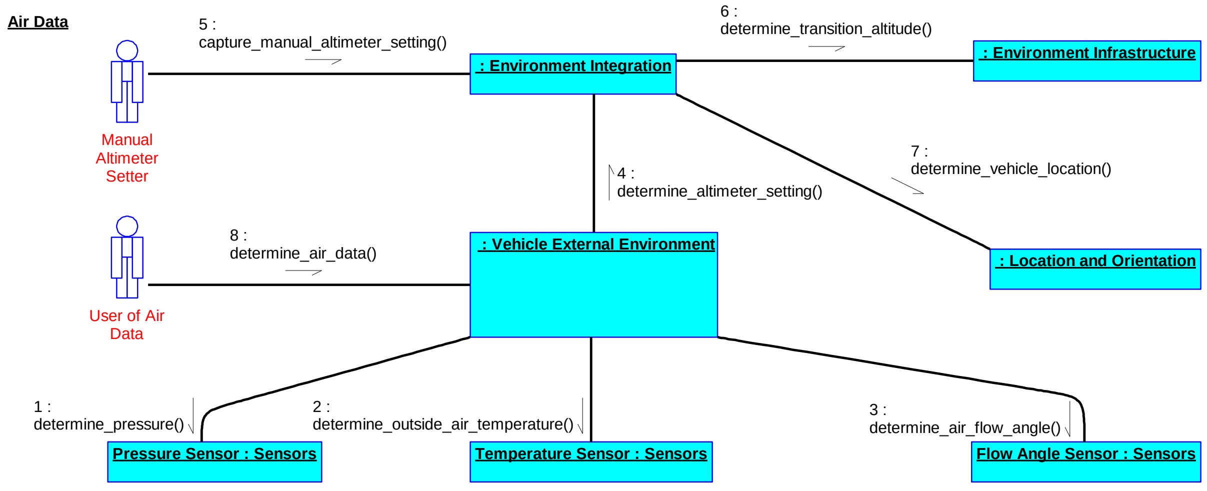

Air Data
5 : capture_manual_altimeter_setting()
6 : determine_transition_altitude()
: Environment Integration
: Environment Infrastructure
Manual Altimeter Setter
7 : determine_vehicle_location()
4 : determine_altimeter_setting()
8 : determine_air_data()
: Vehicle External Environment
: Location and Orientation
User of Air Data
1 : determine_pressure()
2 : determine_outside_air_temperature()
3 : determine_air_flow_angle()
Pressure Sensor : Sensors
Temperature Sensor : Sensors
Flow Angle Sensor : Sensors

Figure 159: Air Data IV

Vehicle External Environment captures values of pressure, temperature and flow angle from instances of Sensors (pressure sensor, temperature sensor and flow angle sensor) to determine air data parameters.

To determine a barometric reference altitude (e.g. referenced to QNH), Vehicle External Environment requests the altimeter setting from Environment Integration. Environment Integration determines the altimeter setting from either a direct input (from Manual Altimeter Setter) or automatically (for an uncrewed vehicle that has lost communications). The automatically determined altimeter setting may be based on the last manual altimeter setting input or transition altitude at the current location (using inputs from Environment Infrastructure and Location and Orientation).

Air data, including accuracy, is provided to the User of Air Data. This could include HMI components for crew displays.

C.2.2.5.3 Post-Conditions

- Up-to-date air data is available.

C.2.2.5.4 Actors

Manual Altimeter Setter

A user directly inputting the altimeter setting, most likely the crew of the vehicle.

The altimeter setting is used to offset pressure altitude so it represents altitude with respect to a particular reference (e.g. 1013.25mb, QNH or QFE).

User of Air Data

Another part of the system, or crew, which uses air data.

C.2.3 Vehicle Stores Views

This section contains IVs relating to the management of stores carried by an air vehicle.

The Vehicle Stores Views shows all the use cases within the Vehicle Stores Views and any actors that interact with them.

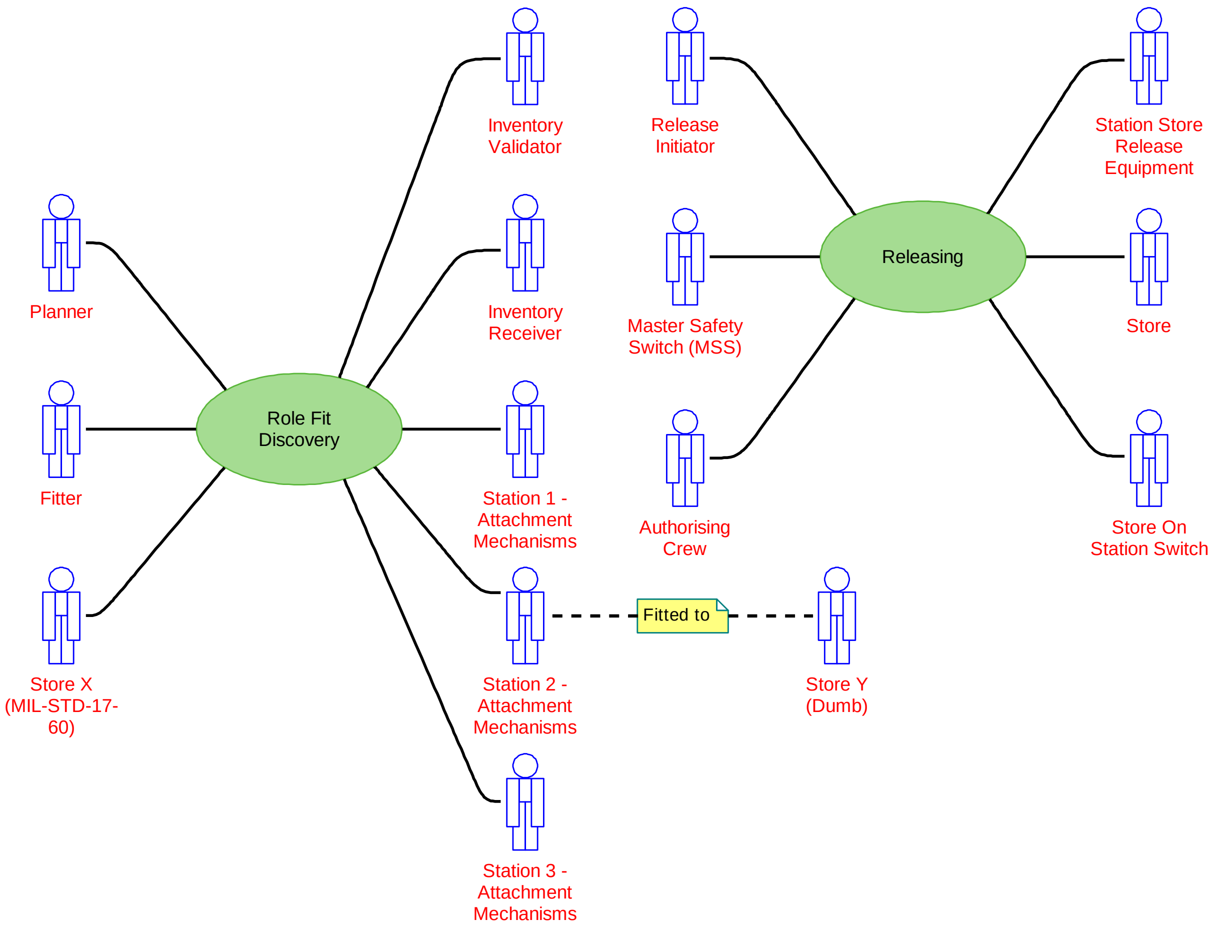


Figure 160: Vehicle Stores Views UC Diagram

C.2.3.1 Role Fit Discovery

This IV shows how role fit equipment fitted to the vehicle is discovered and validated. This scenario describes a system which has 3 stations with the following role fit:

- Station1 - Store X (MIL-STD-1760) Ref. [10].
- Station2 - Store Y (Dumb).
- Station3 - empty.

Each station has a number of attachment mechanisms. The electrical connections at these attachment mechanisms are used to monitor whether a store is connected.

It is assumed that:

- Inventory is provided with the allowable role fit configurations for the vehicle stations.
- The planned and fitted role fit configurations are correct.

- Store X (MIL-STD-1760) communicates via MIL-STD-1760.
- Store Y (Dumb) does not communicate.
- The inventory needs to be validated after passing checks.

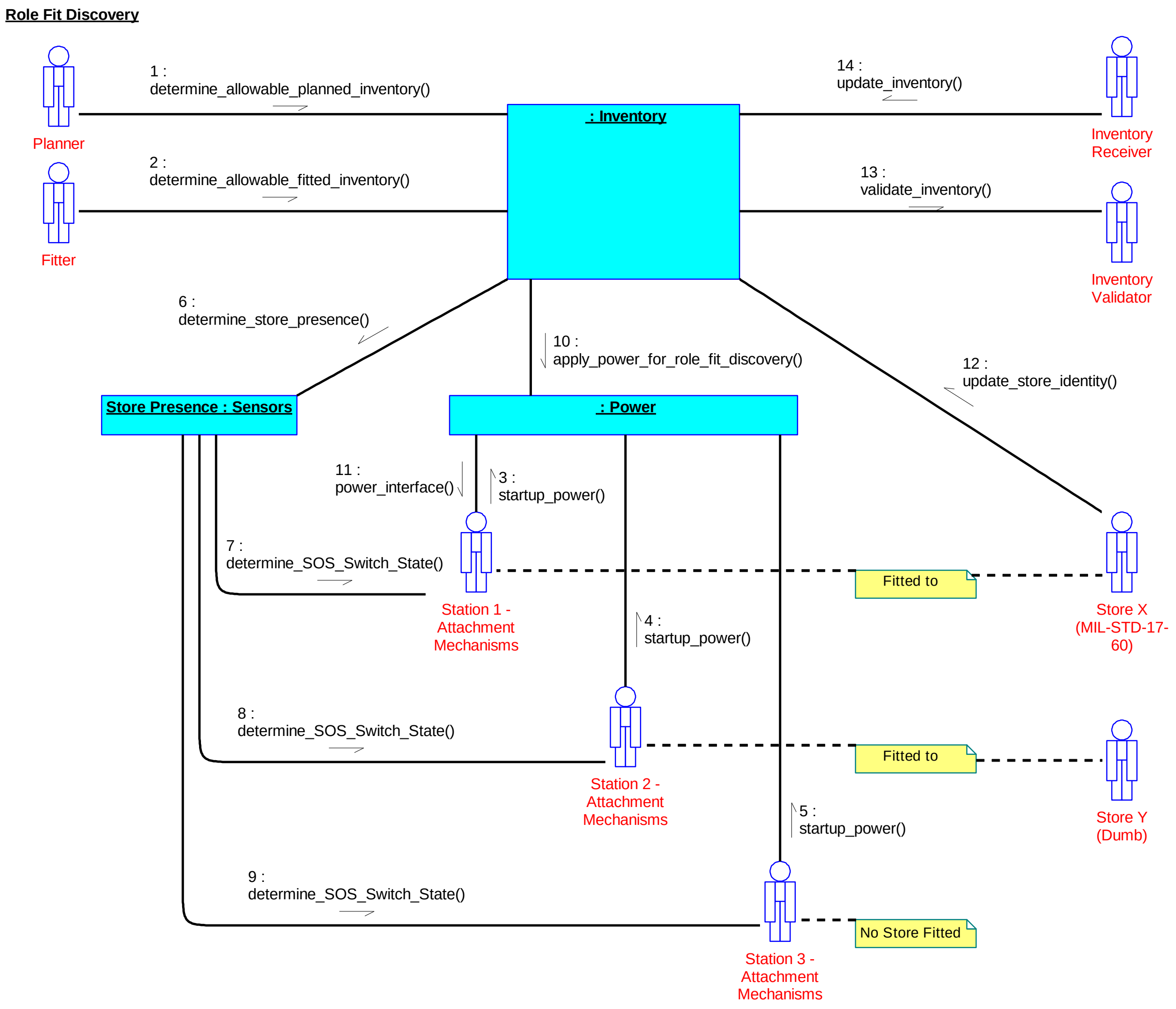
The following considerations are excluded:

- HMI interactions between the Planner, Fitter, the Inventory Validator and certain Inventory Receivers.
- The startup sequence associated with the provision of power.

C.2.3.1.1 Pre-Conditions

- Stores are fitted to stations.

C.2.3.1.2 View



Role Fit Discovery
Planner
Fitter
Inventory Receiver
Inventory Validator
: Inventory
1 : determine_allowable_planned_inventory()
2 : determine_allowable_fitted_inventory()
14 : update_inventory()
13 : validate_inventory()
6 : determine_store_presence()
10 : apply_power_for_role_fit_discovery()
12 : update_store_identity()
Store Presence : Sensors
: Power
11 : power_interface()
3 : startup_power()
4 : startup_power()
5 : startup_power()
7 : determine_SOS_Switch_State()
8 : determine_SOS_Switch_State()
9 : determine_SOS_Switch_State()
Station 1 - Attachment Mechanisms
Station 2 - Attachment Mechanisms
Station 3 - Attachment Mechanisms
Fitted to
Fitted to
No Store Fitted
Store X (MIL-STD-17-60)
Store Y (Dumb)

Figure 161: Role Fit Discovery IV

Upon receipt of the planned and fitted inventories, Inventory determines that both are allowable and that what is stated as fitted matches the planned inventory.

As part of startup, Power will provide limited power to store attachment mechanisms in accordance with the planned and fitted inventory.

Store Presence monitors the state of store on station switches in each attachment mechanism, in this case Station 1 - Attachment Mechanisms and Station 2 - Attachment Mechanisms have stores present and Station 3 - Attachment Mechanisms do not. Inventory then verifies station 1 and station 2 as having items present and station 3 as not having items present, which is consistent with the planned and fitted inventory.

Inventory will request Power to supply specific stores with power to generate electrical signals (in this case store description messaging).

When Power is applied to the store, the store provides its store identity to Inventory. Inventory confirms that the identity of the store at station 1 is as expected. A store identity is not received for station 2 or station 3, since the store fitted to station 2 is dumb and station 3 does not have a store fitted.

Now that Inventory knows which stations have stores present and has identified all of the stores that it can, and has verified these against the fitted, planned and allowable inventories, it presents the verified inventory to the Inventory Validator for validation. Once the inventory is validated, Inventory will continuously provide updates on the current inventory to any Inventory Receiver.

C.2.3.1.3 Post-Conditions

- Inventory Receivers are informed of the current inventory.

C.2.3.1.4 Actors

Fitter

Responsible for defining which inventory was fitted to the system.

Planner

Responsible for defining the planned inventory and inputting it into the system.

Inventory Validator

Responsible for validating the inventory once the Inventory component has performed the necessary checks. This is likely to be a user/operator of the system and it may be possible for them to override the details of some of the inventory before validation.

Inventory Receiver

A generalisation of users of the inventory which require knowledge of the current inventory in order to fulfil their responsibilities.

Examples of users which may need to know the current inventory are:

- For releasable inventory, Stores Release determines what release packages are currently available.
- For external tanks, Fluids determines an increase in the tanks it has and the total contents.
- For a sensor such as a rangefinder, Sensors determines an increase in capability.

Station 1 - Attachment Mechanisms

The set of attachment mechanisms (e.g. clamps) for station 1.

Station 2 - Attachment Mechanisms

The set of attachment mechanisms (e.g. clamps) for station 2.

Station 3 - Attachment Mechanisms

The set of attachment mechanisms (e.g. clamps) for station 3.

Store X (MIL-STD-1760)

A particular store which can communicate via MIL-STD-1760 Ref. [10] protocols.

Store Y (Dumb)

A particular store which cannot communicate (is "Dumb").

C.2.3.2 Releasing

This IV illustrates the armed release of a package of stores when required.

Potential store types include weapons (e.g. AMRAAM, Paveway IV or Brimstone), fuel tanks, countermeasures (e.g. chaff or flares), sensor pods, deployable sensors (e.g. sonobuoys) or gun rounds. This IV covers a rail launched weapon, to illustrate the possible interactions. Other stores would involve different interactions, mostly with the same set of components. A jettison would involve mostly the same set of components, though the physical release mechanism may be different (e.g. jettison of rail launched store may be achieved by downward ejection of the launcher), release intervals may be different and weapon arming would not be activated (or be positively de-activated for enhanced safety).

It is assumed that:

- The weapons bay doors are open.

The following considerations are excluded:

- Controlling the air vehicle to fly within the required flight envelope for release is excluded as this is covered by the Vehicle Movement In Air IV.
- Selection of a store package (types and quantity) that is suitable for a particular task.
- Determining the precise location of the release and the routing to arrive at that location.
- Configuring the air vehicle for release or jettison (e.g. commanding weapon bay door opening).
- Applying logic power and priming of the weapon with, for example, target location.

C.2.3.2.1 Pre-Conditions

- The hardware Master Safety Switch (MSS) (or Master Armaments Safety Switch (MASS) per Def Stan 00-970 Ref. [11] terminology) has already been set to LIVE.

C.2.3.2.2 View

Releasing

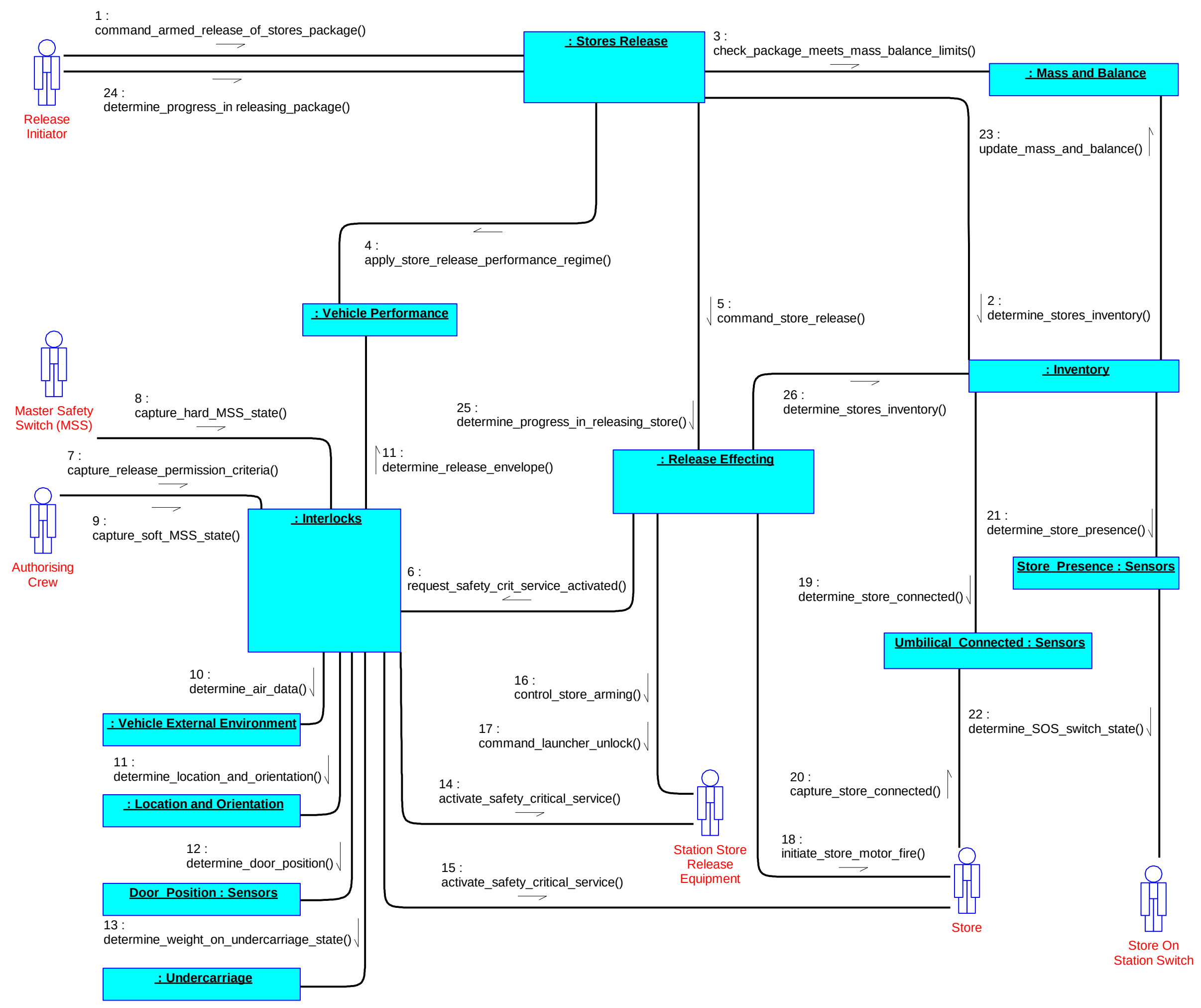

1 : command_armed_release_of_stores_package()
24 : determine_progress_in releasing_package()
Release Initiator
: Stores Release
3 : check_package_meets_mass_balance_limits()
: Mass and Balance
23 : update_mass_and_balance()
4 : apply_store_release_performance_regime()
: Vehicle Performance
5 : command_store_release()
2 : determine_stores_inventory()
: Inventory
Master Safety Switch (MSS)
8 : capture_hard_MSS_state()
25 : determine_progress_in_releasing_store()
26 : determine_stores_inventory()
7 : capture_release_permission_criteria()
11 : determine_release_envelope()
: Release Effecting
Authorising Crew
9 : capture_soft_MSS_state()
: Interlocks
21 : determine_store_presence()
Store Presence : Sensors
6 : request_safety_crit_service_activated()
19 : determine_store_connected()
Umbilical Connected : Sensors
10 : determine_air_data()
16 : control_store_arming()
: Vehicle External Environment
17 : command_launcher_unlock()
22 : determine_SOS_switch_state()
11 : determine_location_and_orientation()
14 : activate_safety_critical_service()
20 : capture_store_connected()
: Location and Orientation
Station Store Release Equipment
12 : determine_door_position()
18 : initiate_store_motor_fire()
15 : activate_safety_critical_service()
Door Position : Sensors
Store
13 : determine_weight_on_undercarriage_state()
Store On Station Switch
: Undercarriage

Figure 162: Releasing IV

When the need for armed release of a store type has been determined, the Release Initiator commands a release. Stores Release confirms which stores are present with Inventory and selects a release package that conforms to Mass and Balance constraints. Whilst the release is in progress Vehicle Performance will be informed of the Stores Release activity, Vehicle Performance will determine the impact of the activity on the performance regime and limit the flight envelope accordingly. Stores Release commands Release Effecting to release the individual store in the package.

Release Effecting requests Interlocks to activate safety critical services for the resources needed to perform the release. Interlocks has knowledge of the interlocks required for a particular safety critical service and captures the release permission criteria (from Authorising Crew) and release envelopes (from Vehicle Performance). Release permission criteria could be in the form of a binary "armed/safe" input, such as that provided by a "Late Arm" switch. Interlocks activates the requested safety critical services provided the appropriate interlocks, release envelope and release permission criteria are satisfied using inputs from Master Safety Switch (MSS) (hard MSS state), Authorising Crew (for "soft" MSS), Vehicle External Environment (for airspeed, etc.), Location and Orientation (for vehicle position and attitude data), Undercarriage (for weight on undercarriage state), and Door_Position: Sensors (for weapon bay door position).

Release Effecting commands Station Store Release Equipment to unlock the rail launched missile and to activate the arming discrete. Once Release Effecting has received confirmation that the missile is unlocked, it commands the stores rocket motor to fire, which will result in the store leaving the aircraft.

During the release sequence Inventory will update the role fit inventory based on inputs from Umbilical_Connected: Sensors (store umbilical connected) and Store_Presence: Sensors which monitors Store On Station Switch. Mass and Balance will reflect the revised stores configuration.

Stores Release provides progress on the requirement to release the stores to Release Initiator based on progress reports of individual releases from Release Effecting.

C.2.3.2.3 Post-Conditions

- Store has been released.
- The inventory, mass and balance have been updated.

C.2.3.2.4 Actors

Authorising Crew

The crew that are authorising the release and controlling the "soft" Master Safety Switch (MSS).

Master Safety Switch (MSS)

The hardware switch located on the air vehicle. It is normally LIVE throughout take-off, flight and landing.

Release Initiator

The Release Initiator determines the store type (e.g. ASRAAM) to be released and initiates the release. Initiation could be from Action components (e.g. Target Engagement) or from the crew (via HMI).

Station Store Release Equipment

The equipment that directly outputs electrical discretes (release, fusing, etc.) to S&RE or stores if commanded to do so and the appropriate safety critical service is enabled. Typically this equipment will include power drivers and relays.

Store

A store fitted to a particular station.

Store On Station Switch

A Store On Station (SOS) switch.

C.2.4 Communications Views

This section contains IVs relating to the communications capability of a system.

The Figure 163: Communication Views UC Diagram shows all the use cases within the Communications Views and any actors that interact with them.

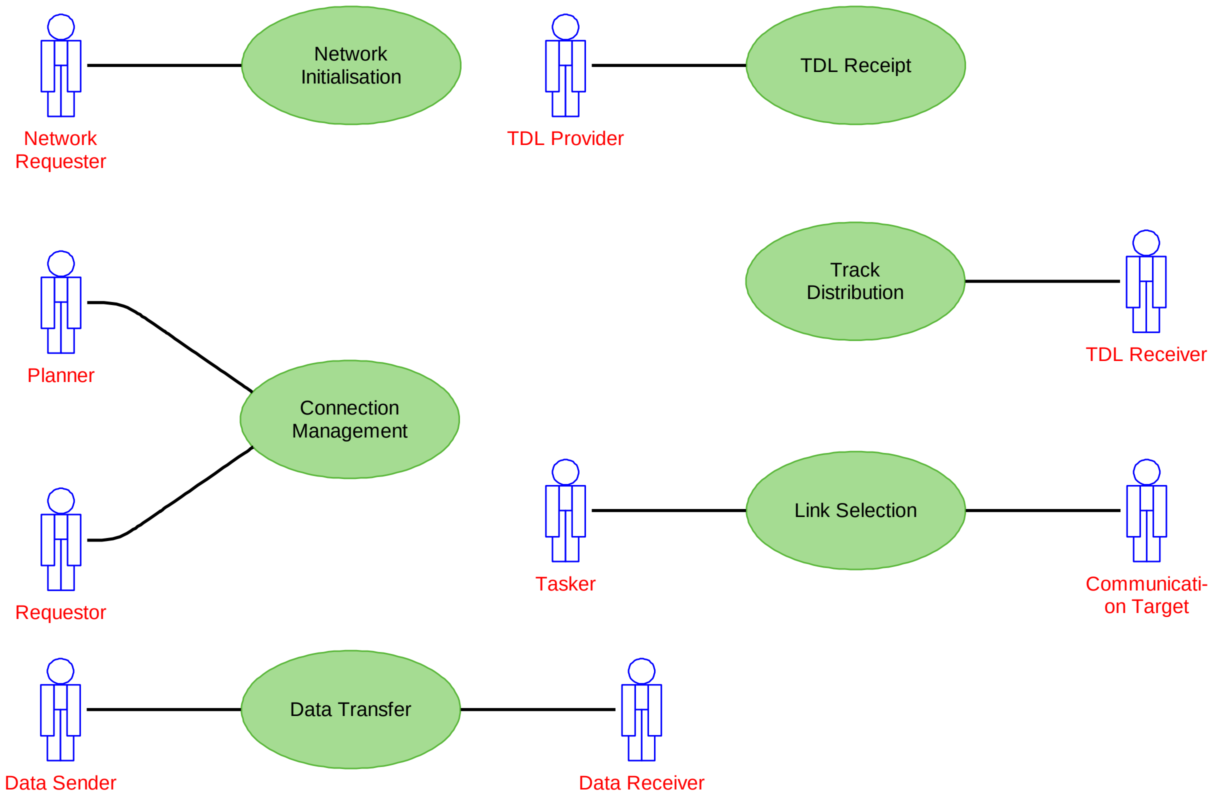


Figure 163: Communication Views UC Diagram

C.2.4.1 Network Initialisation

This IV illustrates how the system satisfies a requirement to set up a series of communication links to form a new network (either for use immediately or in the future) to support one or more tasks.

It is assumed that:

- No existing network route exists which can provide the support required.

The following considerations are excluded:

- Use of the network once established.
- The details of data transfer over the network.
- The detailed control of a Communicator.
- Harmonisation of antennae in flight via Spatial Correction.
- Coordination of multiple activities across the EM spectrum.
- Detailed use of cryptography.
- The brokering of power.

C.2.4.1.1 Pre-Conditions

- A task that requires communications over a network exists.

C.2.4.1.2 View

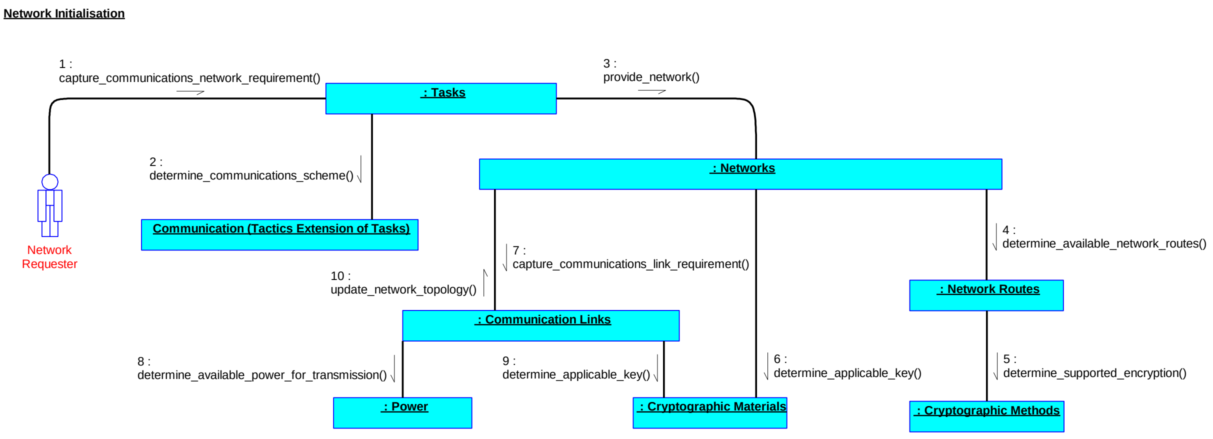

Network Initialisation
1 : capture_communications_network_requirement()
: Tasks
3 : provide_network()
2 : determine_communications_scheme()
: Networks
Network Requester
Communication (Tactics Extension of Tasks)
4 : determine_available_network_routes()
7 : capture_communications_link_requirement()
10 : update_network_topology()
: Network Routes
: Communication Links
8 : determine_available_power_for_transmission()
9 : determine_applicable_key()
6 : determine_applicable_key()
5 : determine_supported_encryption()
: Power
: Cryptographic Materials
: Cryptographic Methods

Figure 164: Network Initialisation IV

Tasks, supported by the Communication tactics extension, captures a requirement for communications from a Network Requester, and instructs Networks to provide the required communications network. Networks, Network Routes, Cryptographic Methods and Cryptographic Materials determine that the need for communications cannot be met by existing network routes, and cannot be met to a required cryptographic standard. Hence a new communications link is required.

Networks determines the requirements for this new link, and the requirements are captured by Communication Links.

Communication Links confirms with Power and Cryptographic Materials the availability of the required resources. Communication Links informs Networks that the required communication link can be established and Networks updates its topology accordingly.

C.2.4.1.3 Post-Conditions

- A communications link is fully planned and agreed for transmitting and receiving communications capabilities, and can either be used immediately or at a later point in time to support the tasks illustrated in the Human Communications and Data Transfer IVs.

C.2.4.1.4 Actors

Network Requester

A component that has determined there is a need for additional communications infrastructure to be provided.

C.2.4.2 Connection Management

This IV shows how communications can be planned, and later established, to meet a given communication requirement. This includes showing how continual feedback from network resources is used to aid prediction of communication achievability.

A Planner is aware of a communication requirement and needs to assess whether the resources will be available to meet the demand; taking into consideration reported performance data. It is determined that the communication requirement is achievable at a planned time. When the communications are needed, a Requestor makes a demand to establish communications.

The interactions shown are applicable whether or not the Planner and Requestor are the same individual or entity. Likewise, it remains the same, when the communication requirement is ad-hoc for immediate use or planned for a later time.

The following considerations are excluded:

- Control of communications infrastructure.
- Control of the communicator.
- Observability analysis.
- Use of cryptography.
- Permitted transmission capabilities.
- Use of the communication to transfer data.

C.2.4.2.1 Pre-Conditions

- The nodes making up the network are known; either entered during prior planning or by data configuration.
- The preferred transfer solutions for different types of data for different recipients have been preloaded in the relevant components.

C.2.4.2.2 View

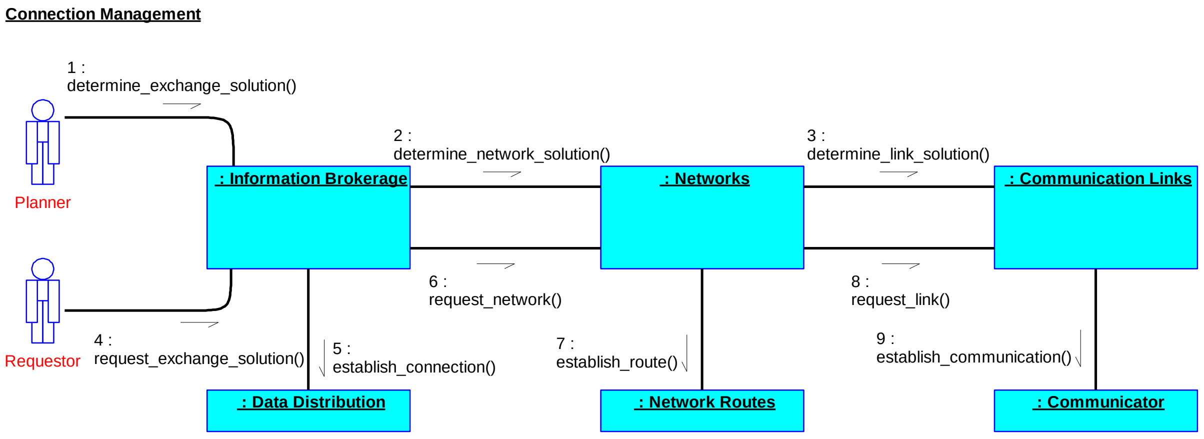


Figure 165: Connection Management IV

A Planner asks Information Brokerage to determine an information exchange solution.

Information Brokerage determines that the exchange is allowed and, as this is a request for a new data exchange of a known type, checks with Networks for the expected reachability across the network.

Networks evaluates possible solutions, including determining the quality of service required to meet the connectivity preferences. As there is currently no network in place Networks requests recommended link solutions from Communication Links.

Based on the request from Networks, Communication Links checks for the best link solution to meet the latency, reliability and integrity requirements. It also considers the real-world constraints on communication between vehicles, this includes support from Observability (not shown).

Communication Links determines that the link requirement is achievable. With this information Networks can choose a network solution, informing Information Brokerage that communication requirement is achievable. Based on this feedback Information Brokerage determines that the information exchange is achievable and informs the Planner.

Throughout this process, and in support of all communications, communication resources provide continuous feedback which is analysed by the communications components; this allows achievability of communication requirements to be continuously assessed and remedial actions to be performed to inform achievability determination in future service requests.

At a later time a Requestor requests that Information Brokerage establish the previously planned information exchange service. Information Brokerage asks Data Distribution to facilitate data exchange, in line with the service requirements, and requests establishment of the network from Networks.

Networks in turn asks Network Routes to configure routes, in line with the routing requirements, and requests Communication Links to establish links.

Communication Links asks the Communicator to establish communication, in-line with the communication requirements.

C.2.4.2.3 Post-Conditions

- Communications resources have been configured to meet the communication requirements.

C.2.4.2.4 Actors

Planner

A component that has communication requirements which need to be planned, either for immediate or later use.

Requestor

A component that has communication requirements which need to be established, for immediate or later use.

C.2.4.3 Data Transfer

This IV shows how data is transferred to a recipient across a communications link, including both transmission and reception. The monitoring of the data transfer service is also shown, to highlight the management components' responsibilities for maintaining the quality of the communication service.

Note: Cryptographic Methods is shown as a service used by Network Routes; however at a platform level, for security reasons, Cryptographic Methods may be in-line, between Network Routes and Communicator.

It is assumed that:

- Only packet encryption is required for the data transfer.

The following considerations are excluded:

- How the data transfer request is made.
- How data is passed between the transmitting and receiving nodes.
- How the service provision dynamically changes in response to quality of service monitoring.

C.2.4.3.1 Pre-Conditions

- All necessary communications services have been set up and established, in order to support interactions between data senders and communications transport.
- Transmission protocol has been determined.
- Encryption requirements have been determined.
- The data to be transferred exists within the system.
- The transfer of data has been initiated.

C.2.4.3.2 View

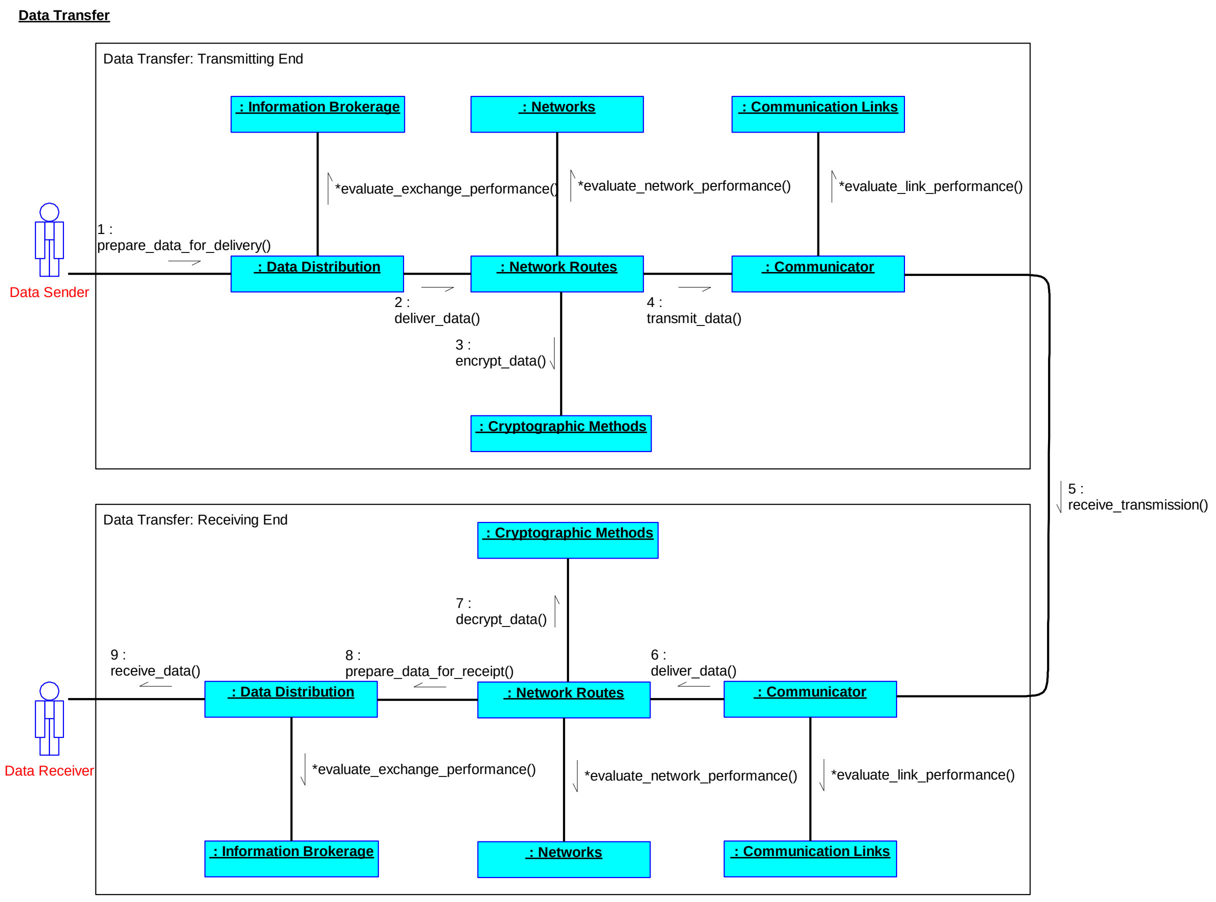


Figure 166: Data Transfer IV

Transmission

Data Distribution prepares the data for delivery by formatting the data according to the transmission protocol (determined during communications planning); this may involve splitting the data into multiple packets. Network Routes directs the prepared packets for forward data delivery. Network Routes determines which route the data will take. This may include going via Cryptographic Methods to encrypt the packets prior to transmission, depending upon the security policy determined during communications planning. Actual transmission of data is performed by Communicator.

Reception

Communicator receives incoming data which it passes, as packets, to Network Routes for determination of the next route to take (in this case to report to the local system, as the data has arrived at its destination). Network Routes may use Cryptographic Methods to decrypt the packets prior to delivery, depending upon the communications setup. The packets are given to Data Distribution in order that they can be prepared for use. This involves collating the packets into a single data source and reformatting the data. When the data has been prepared, a receive request is issued to pass the completed data to its intended recipient.

At all times, the Resource Layer components (Network Routes and Communicator) report on the performance of the service they provide, in order that the Action Layer components can evaluate performance and determine whether resources need reconfiguring to maintain the optimum level of service.

C.2.4.3.3 Post-Conditions

- The intended end user has the transmitted data.

C.2.4.3.4 Actors

Data Receiver

The intended recipient of the data.

Data Sender

The provider of the data to be transferred. This could be a human initiated request, or an automated transfer.

C.2.4.4 Tactical Exchange

C.2.4.4.1 Track Distribution

This IV shows how a track is distributed to an external system via a tactical datalink. This IV is specific to exchanging tracks via a tactical datalink.

It is assumed that:

- The deployment is set up so the components can access the track when required.

The following related areas of functionality are excluded:

- How tracks are produced from sensor data.
- How tactical datalink connections are managed.
- Non-track exchanges via tactical datalink.

C.2.4.4.1.1 Pre-Conditions

- Information Brokerage has been loaded with the rules around what is allowed to be distributed outside of the system.
- The rules on how to declassify a track are loaded to Information Brokerage.
- The connection type that Data Distribution needs to use for the external system and the rules for initiating an exchange has been previously loaded.
- Tactical datalink structure rules are loaded to Semantic Translation.
- Tactical datalink delivery rules are loaded to Data Distribution.

C.2.4.4.1.2 View

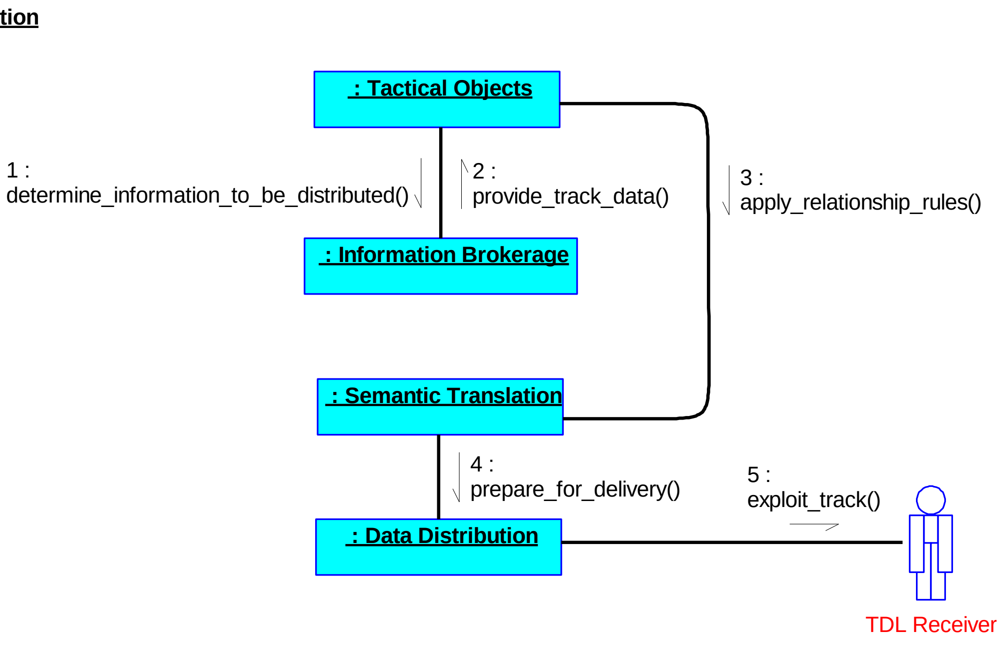


Figure 167: Track Distribution IV

Tactical Objects produces a track set. Information Brokerage determines that information within the track set meets a distribution requirement of an external system.

Information Brokerage understands that to distribute the track to the specific external system it will need to be exchanged via a tactical datalink. This includes determining that the system has reporting responsibility for the track and that only some of the track attributes should be shared for classification reasons. Information Brokerage directs the release of the data by Tactical Objects. Information Brokerage does not handle the data directly.

Tactical Objects provides the data to Semantic Translation for distribution to the external system. Semantic Translation receives the required track information from Tactical Objects, applies the necessary translation rules to the track data and reformats it into the tactical datalink structure, ensuring the correct structure is in place before the distribution is initiated.

Once the track data has been reformatted, Data Distribution initiates distribution to the TDL Receiver in accordance with established rules. Data Distribution does this by determining the communication protocol and packages data into messages as defined by the protocol. Data Distribution then distributes the track to the TDL Receiver for exploitation.

C.2.4.4.1.3 Post-Conditions

- A track has been distributed via a tactical datalink.

C.2.4.4.1.4 Actors

TDL Receiver

A receiver capable of receiving tracks from a tactical datalink system.

C.2.4.4.2 TDL Receipt

This IV shows how a message is received from an external system that follows a different logic to the local system. In this scenario a TDL system sends a message, which is an overloaded message type and is referencing different subject matters. This is resolved by Semantic Translation which converts the TDL specific logic into something understandable by the local system.

It is assumed that:

- The deployment is set up so the components can access the related information.

The following related areas of functionality are excluded:

- How tactical datalink connections are managed.
- Further analysis (including data fusion) of received TDL information.
- Low level data exchange using the Communicator component.

C.2.4.4.2.1 Pre-Conditions

- The connection type that Semantic Translation needs to use for the external system has been previously loaded.
- Tactical datalink structure rules are loaded to Semantic Translation.
- Tactical datalink receipt rules are loaded to Data Distribution.
- Ownship has conducted an engagement of a target using remote track data received over Link-16 tactical datalink.

C.2.4.4.2.2 View

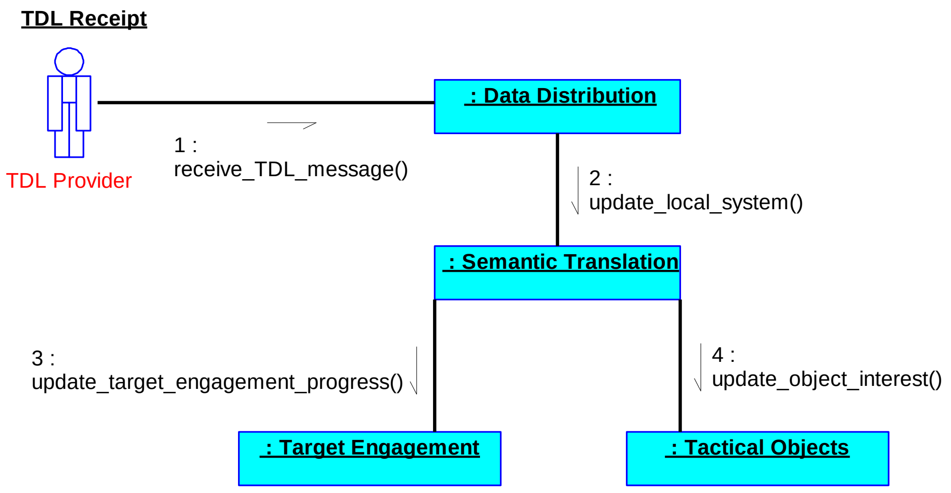

TDL Receipt
TDL Provider
1 : receive_TDL_message()
: Data Distribution
2 : update_local_system()
: Semantic Translation
3 : update_target_engagement_progress()
4 : update_object_interest()
: Target Engagement
: Tactical Objects

Figure 168: TDL Receipt IV

TDL Provider provides a '*target sorting'* message (J12.6) from a Link-16 tactical datalink to Data Distribution. Data Distribution then unpacks the message into the individual status information discretes. As these are tactical datalink specific they are passed to Semantic Translation for conversion for use by PYRAMID components.

Semantic Translation analyses the information from the external system and identifies that it is a destruction of target confirmation. Using its mapping rules Semantic Translation determines the related internal events. Target Engagement is informed of the updated engagement result, and Tactical Objects is informed of the change in reported status of the referenced object.

C.2.4.4.2.3 Post-Conditions

- The relevant engagement update has been captured.
- The relevant object has been updated.

C.2.4.4.2.4 Actors

TDL Provider

A provider that has non-C2 reporting responsibility within a tactical datalink system.

C.2.4.5 Link Selection

This IV shows how the most appropriate communication solutions can be utilised as circumstances change. In this case, it is shown how a HF communications link with a Communication Target is established and then, as the communications distance reduces, a UHF link is established as a more effective solution.

It is assumed that:

- The radio supports a signal lock capability, allowing local control of the antenna to maintain the best signal.

The following considerations are excluded:

- Mechanical steering.
- Determination of own position.
- Determination of Communication Target position (this varies depending upon the method used to determine the position).
- Flow of data for communication.
- Translation between target location and pointing directions.
- EM interoperability, e.g. blanking.
- Harmonisation via Spatial Correction.
- Planning of communications, e.g. detailed considerations of how and when to make selection changes.
- Measurement of communications quality.

C.2.4.5.1 Pre-Conditions

- All necessary communications services have been set up and established sufficiently to support link management.
- The location of the Communication Target is known.
- HF and UHF resources are available at both ends.

C.2.4.5.2 View

Link Selection

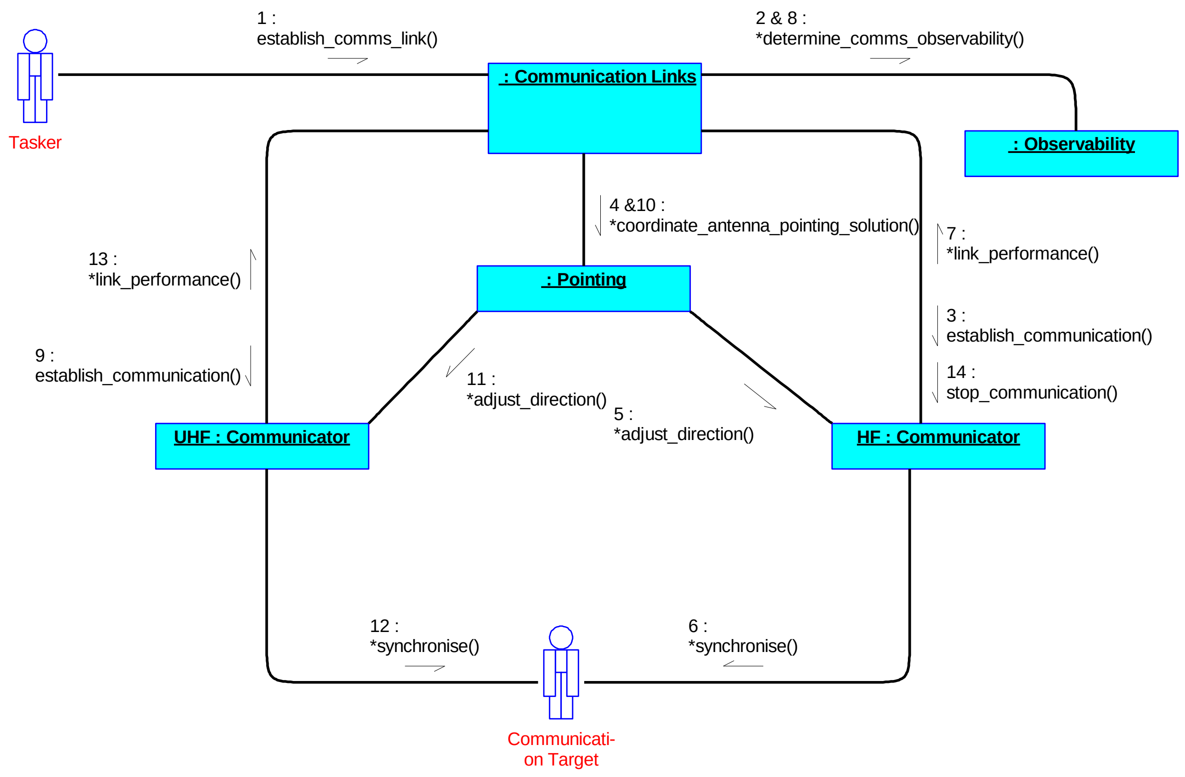


Figure 169: Link Selection IV

Tasker requests the establishment of communications from Communication Links. Communication Links determines the costs of possible links based on observability information from Observability. Using this, alongside knowledge of the supported options for connection to the Communication Target, Communication Links determines that a HF radio link is the best solution as the range is greater than can be achieved via UHF communications.

Communication Links commands the HF Communicator to start establishing communications with the Communication Target. Communication Links commands Pointing to point the HF Communicator at the known location of the Communication Target; Pointing directs the HF Communicator to electronically course steer by providing directional instructions.

The HF Communicator synchronises its connection with that of the Communication Target, setting up the communications channel. This synchronisation continues throughout communications, in order to maintain the connection in accordance with the link protocol, with HF Communicator automatically performing fine tuning to maintain the best signal.

Throughout communication, the Communicator reports link performance information to Communication Links. This information, along with Observability results, is used to continually assess if, and when, an alternative communication solution should be selected.

In this case, it is observed that the distance to the Communication Target has reduced, and it is determined that an UHF connection will now better satisfy the communication quality requirements than the existing HF link.

Communication Links, commands UHF Communicator to start establishing communications with the Communication Target. Communication Links commands Pointing to point the UHF Communicator at the known location of the Communication Target; Pointing directs the UHF Communicator to electronically course steer by providing directional instructions.

The UHF Communicator synchronises its connection with that of the Communication Target, setting up the communications channel. Once the alternative link is established Communication Links commands the HF Communicator to stop communicating.

C.2.4.5.3 Post-Conditions

- Communications are established.

C.2.4.5.4 Actors

Communication Target

An external entity with reception capabilities that is the target for communications (such as another instance of Communicator).

Tasker

Higher level system component responsible for determining if, and when, a communication link with a Communication Target is required (e.g. the Tasks or Networks component).

C.2.5 Sensing Views

This section contains IVs relating to the sensing capability of a system.

The Figure 170: Sensing Views UC Diagram shows all the use cases within the Sensing Views and any actors that interact with them.

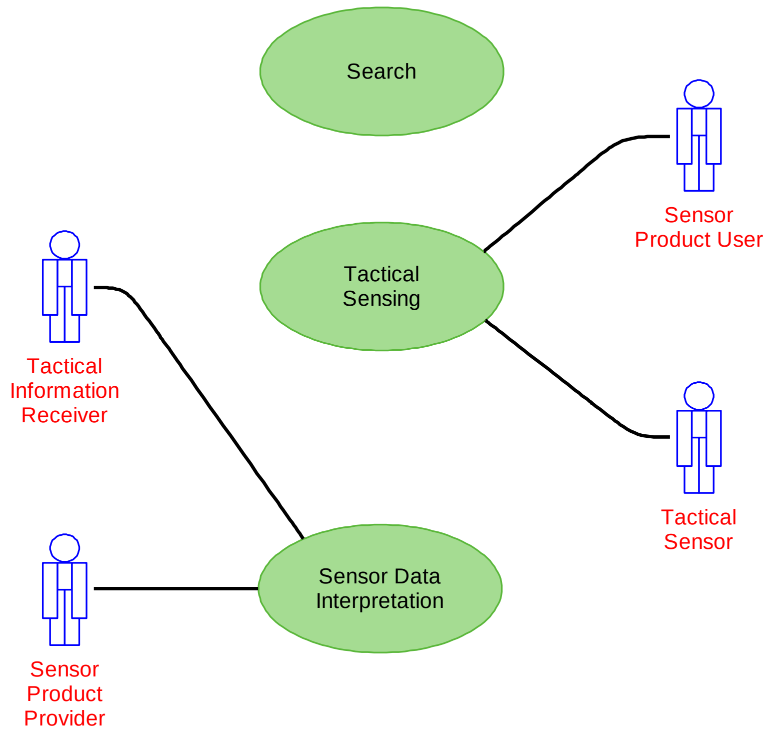


Figure 170: Sensing Views UC Diagram

C.2.5.1 Search

A task has been generated that requires the system to search for a tank within a specified area. This IV illustrates how the task is planned and coordinated using the Task and Action layers, detailed in the Control Architecture PYRAMID concept.

It is assumed that:

- Resources are available when required.
- A single air vehicle is being used.
- A range of sensors are fitted onto the air vehicle.
- There are no limits on the mission (e.g. RoE, EMCON or restricted airspace).

The following considerations are excluded:

- The operation of the sensor to perform the sensing actions.
- The interpretation of captured sensor data.
- The coordination of the routing solution.

C.2.5.1.1 Pre-Conditions

- The air vehicle is en route to an area within its sensors range of the target.
- A task has been issued to search for a tank in a specific region.

C.2.5.1.2 View

Search

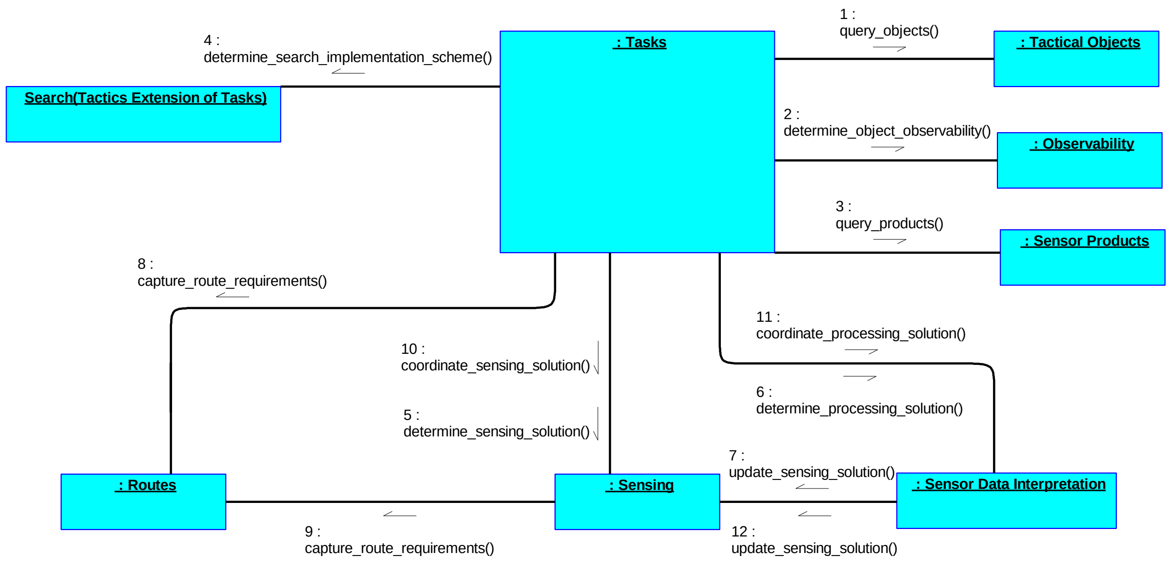

: Tasks
: Tactical Objects
: Observability
: Sensor Products
Search(Tactics Extension of Tasks)
: Routes
: Sensing
: Sensor Data Interpretation
1 : query_objects()
2 : determine_object_observability()
3 : query_products()
4 : determine_search_implementation_scheme()
5 : determine_sensing_solution()
6 : determine_processing_solution()
7 : update_sensing_solution()
8 : capture_route_requirements()
9 : capture_route_requirements()
10 : coordinate_sensing_solution()
11 : coordinate_processing_solution()
12 : update_sensing_solution()

Figure 171: Search IV

Having received the task to search for a tank in a region, Tasks utilises its Search extension to determine if the task can be satisfied using any pre-existing data. Search determines that no pre-existing data exists that satisfies the requirements of the task and that additional data is needed. Search needs to determine the best tactic to use to gather and process the necessary data. In order for it to make these determinations, it queries a number of service components which provide relevant information as follows:

- Tactical Objects is queried for objects with a type that equals tank and location within the specified area, currently there are no objects known which match that criteria.
- Observability is asked for probabilities of making observations of tanks in that region based on the current environmental conditions (e.g. weather conditions). This provides the observation types (e.g. visual, electromagnetic, and thermal) which could be made of a tank and the probabilities of each, since no current location of a tank is known within the region.
- Sensor Products is then queried for sensor data availability matching those observation types covering that region: there are currently no matching products therefore additional sensor coverage needs to be established to capture them.

In addition to this information the capabilities of action components are provided to Tasks. The Search extension uses all this information to determine the tactic to utilise and creates an implementation scheme for this task. A number of actions are defined to facilitate the task along with relevant constraints:

- Sensing is told to generate sensor products of the most probable observation type within the region, and Sensor Data Interpretation is given an action to process these products to identify tanks.
- Sensing and Sensor Data Interpretation then determine the solutions to their actions.
- Sensor Data Interpretation places requirements on Sensing, which refines the requirement given by Tasks.
- Tasks places a requirement on Routes for the air vehicle to operate within the defined search region.
- Sensing places a requirement on Routes for the required positioning of the air vehicle to carry out the search, which refines the requirement given by Tasks.
- With an achievable routing solution and achievable Sensing actions, Tasks commands the execution of the task.

During execution, Sensor Data Interpretation determines that the sensor product is not of sufficient quality to satisfy the requirements of its processing solution (e.g. an image is not of sufficient resolution to find tanks). To deal with this Sensor Data Interpretation updates the requirements on Sensing to increase the quality of the sensor product it produces. Sensing then refines the solution it provides (e.g. chooses another sensor mode) within the constraints given by Tasks.

C.2.5.1.3 Post-Conditions

- The search task is complete.

C.2.5.2 Tactical Sensing

This IV illustrates the enactment of a sensing task in order to produce a sensor product from a single Tactical Sensor. This includes the application of power, and the manoeuvring of the sensor in order to maintain an effective pointing angle.

It is assumed that:

- The Tactical Sensor is externally mounted to the vehicle on a gimbal.
- Application of power to the Tactical Sensor is a pre-requisite to enable access to the capability of the sensor rather than being part of the sensor capability itself.

The following considerations are excluded:

- The planning of a sensing task.
- The processing of captured sensor data.
- The brokering of power.

C.2.5.2.1 Pre-Conditions

- The air vehicle is on route to be within its sensor range of the target.

C.2.5.2.2 View

Tactical Sensing

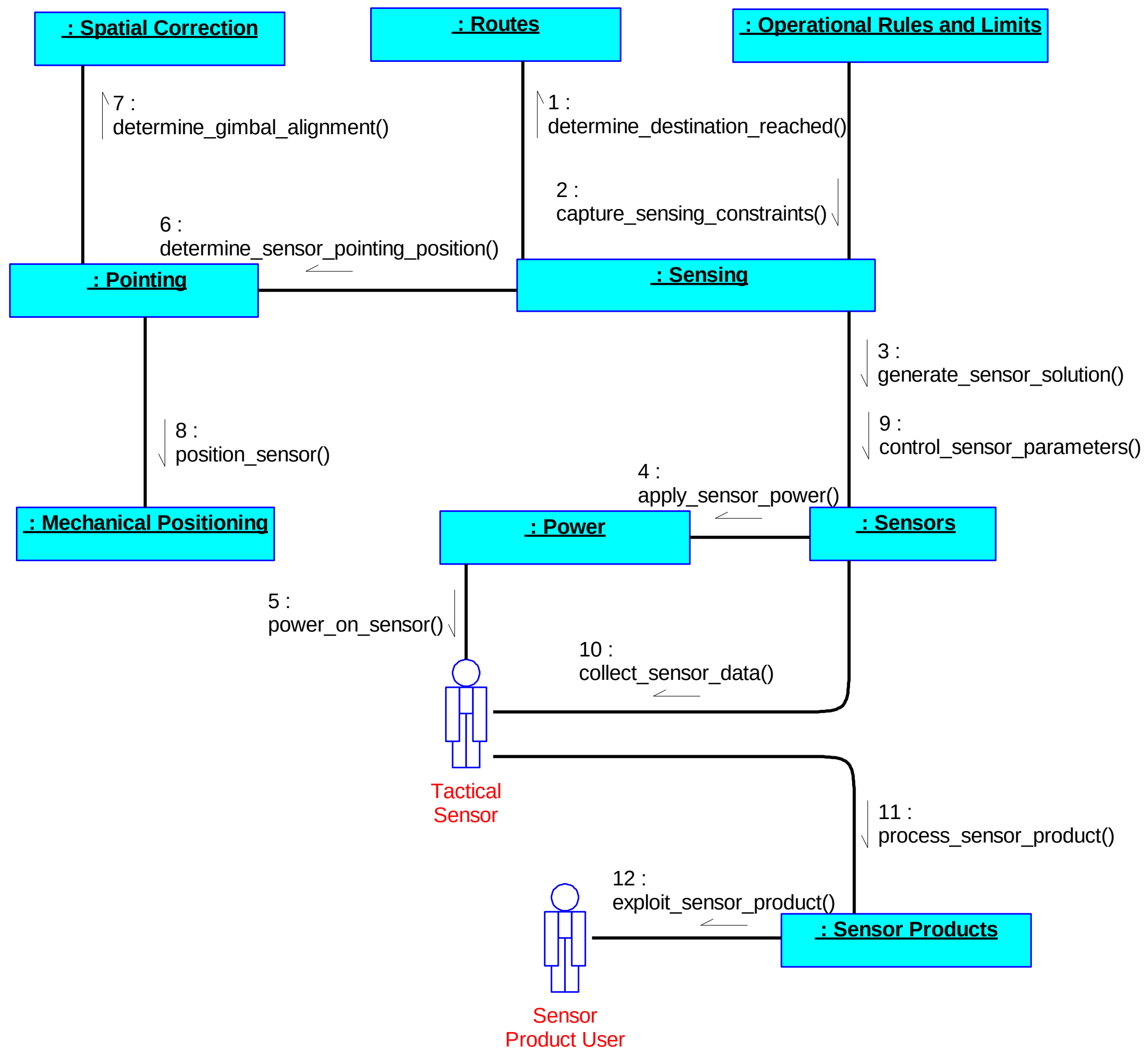

: Spatial Correction
: Routes
: Operational Rules and Limits
7 : determine_gimbal_alignment()
1 : determine_destination_reached()
2 : capture_sensing_constraints()
6 : determine_sensor_pointing_position()
: Pointing
: Sensing
3 : generate_sensor_solution()
9 : control_sensor_parameters()
8 : position_sensor()
4 : apply_sensor_power()
: Mechanical Positioning
: Power
: Sensors
5 : power_on_sensor()
10 : collect_sensor_data()
Tactical Sensor
11 : process_sensor_product()
12 : exploit_sensor_product()
: Sensor Products
Sensor Product User

Figure 172: Tactical Sensing IV

Sensing checks with Routes that the location for the sensing task has been reached, triggering Sensing to begin enacting the coordination of sensing activities.

Operating within the constraints set by Operational Rules and Limits, Sensing informs Sensors to generate the sensor solution to complete the sensing tasks. Sensors ensures power is applied to the Tactical Sensor via Power, while Sensing ensures that the Tactical Sensor is positioned correctly via Pointing, Spatial Correction and Mechanical Positioning. Sensing commands Sensors to apply the relevant modes, settings and demands onto the Tactical Sensor in order to execute the sensing actions.

Sensor data is provided to Sensor Products for processing, until the sensing requirement has been fulfilled, and the resulting sensor product is distributed to any relevant Sensor Product Users.

C.2.5.2.3 Post-Conditions

- A sensor product has been generated for exploitation by a Sensor Product User.

C.2.5.2.4 Actors

Sensor Product User

Any element, either internal or external to the system, that utilises sensor products.

Tactical Sensor

The equipment of any sensor used for tactical aims (e.g. radar, IRST, or EO camera).

C.2.5.3 Sensor Data Interpretation

A task has been planned to maintain situation awareness of objects within the immediate battlespace around the ownship. Tasks is responsible for coordinating components required to execute the task solution. This IV shows the task being executed, leading to the generation of objects within Tactical Objects and the provision of object information by Tactical Objects to Tactical Information Receivers.

It is assumed that:

- The sensor selected as part of the sensing solution is only being used for this task.

The following related areas of functionality are excluded:

- Exchange of tactical information with an external system.
- The execution of sensing activities to generate sensor products.
- The planning of a sensing task.
- Determination of own air vehicle location and kinematics from sensors.
- Sanitisation of an area (interpretation of data to determine confidence of a lack of objects in an area).

C.2.5.3.1 Pre-Conditions

- A Sensor Data Interpretation solution and a Sensing solution have been planned.
- The Sensor Products and Data Fusion components are configured with necessary algorithms.
- The ownship location is known.

C.2.5.3.2 View

Sensor Data Interpretation

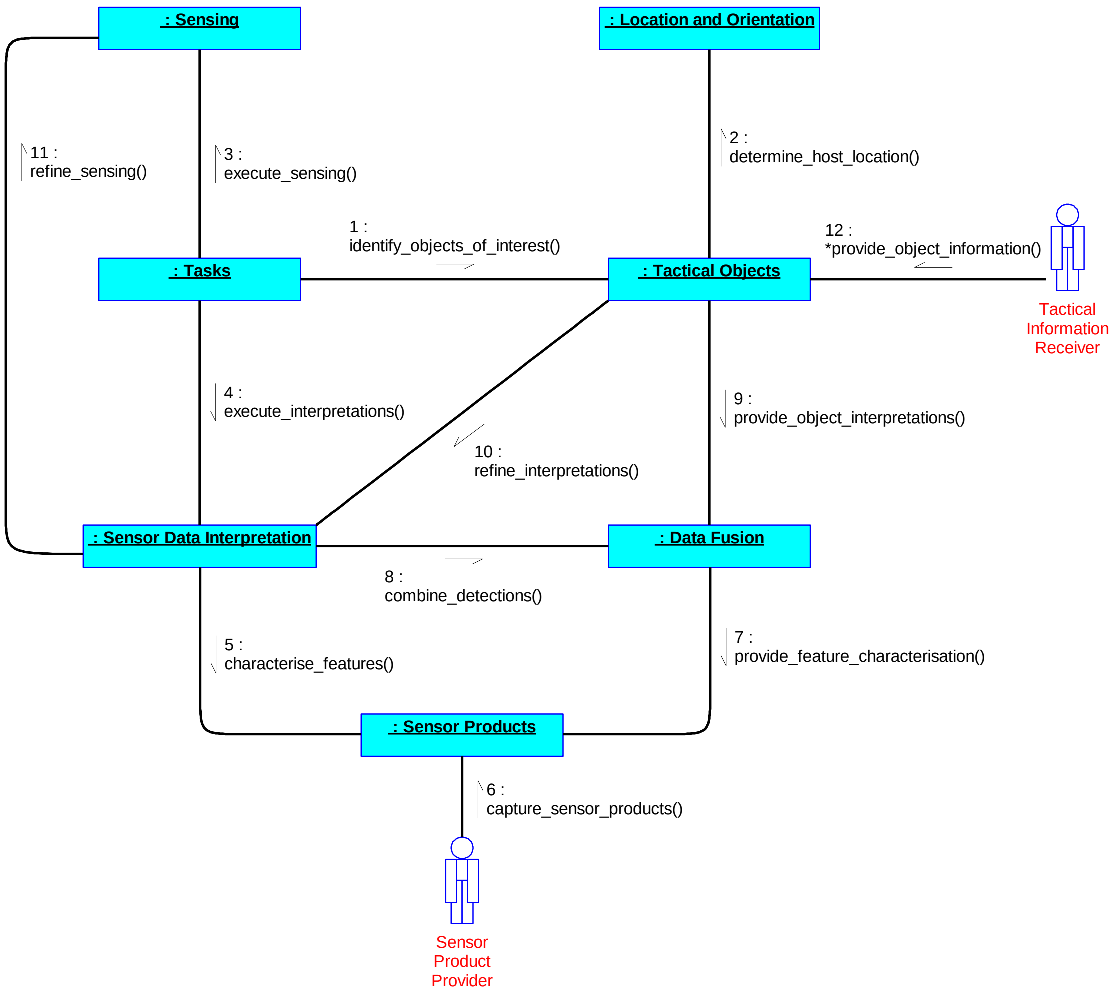

: Sensing
: Location and Orientation
11 : refine_sensing()
3 : execute_sensing()
2 : determine_host_location()
1 : identify_objects_of_interest()
12 : *provide_object_information()
: Tasks
: Tactical Objects
Tactical Information Receiver
4 : execute_interpretations()
9 : provide_object_interpretations()
10 : refine_interpretations()
: Sensor Data Interpretation
: Data Fusion
8 : combine_detections()
5 : characterise_features()
7 : provide_feature_characterisation()
: Sensor Products
6 : capture_sensor_products()
Sensor Product Provider

Figure 173: Sensor Data Interpretation IV

To initiate execution of the task, Tasks places a requirement on Tactical Objects to identify all objects of interest within the region surrounding the air vehicle. As part of this, the characteristics of the objects that are of interest (e.g. position and classification) and the desired level of quality of these characteristics are defined. Tactical Objects requests ownship location from Location and Orientation. Tasks also commands the execution of the planned sensing and interpretation solutions.

For the interpretation solution, Sensor Data Interpretation commands Sensor Products to identify and characterise features from captured sensor data, and Data Fusion to combine those detections to generate interpretations of objects.

Sensor Data Interpretation continuously monitors quality and progress against the requirement and initiates any remedial action.

These processes lead to the generation of objects within Tactical Objects. Based on the requirement it has been given, Tactical Objects will identify dependencies on Sensor Data Interpretation in order to maintain and gather more information about particular objects. Two examples where this could occur are:

- If an object of unknown classification is identified within the region of interest, Tactical Objects will identify a dependency to determine the classification of that object.
- If an object which has been deemed of interest is not of the required quality or begins dropping in quality, Tactical Objects will identify a dependency to increase the quality of that object.

Based on the dependencies identified, Sensor Data Interpretation will refine the interpretation solution (e.g. to process a new type of sensor product or apply new processes) to maintain information or gather additional information about the objects specified in those dependencies.

If required, Sensor Data Interpretation will identify dependencies for new or improved sensor data, which will lead Sensing to refine the sensing solution (e.g. use more sensor resources or focus sensing onto a particular object) to provide that data.

Throughout this scenario, object information will be provided by Tactical Objects to Tactical Information Receivers.

C.2.5.3.3 Post-Conditions

- Tactical Information Receivers have knowledge of objects surrounding the vehicle.

C.2.5.3.4 Actors

Sensor Product Provider

An entity which can provide sensor products to the system, this could be a particular tactical sensor (e.g. radar).

Tactical Information Receiver

An entity which has an interest in tactical objects and the information about them to fulfil its own responsibilities. Examples of this could be other tactical information components (e.g. Threats or Susceptibility) or entities which need to make decisions which are based on tactical information such as an authorised operator or the Tasks component.

C.2.6 Support Functions Views

This section contains interaction views relating to functions that support the operational use of a system.

The Figure 174: Support Functions Views UC Diagram shows all the use cases within the Support Functions Views and any actors that interact with them.

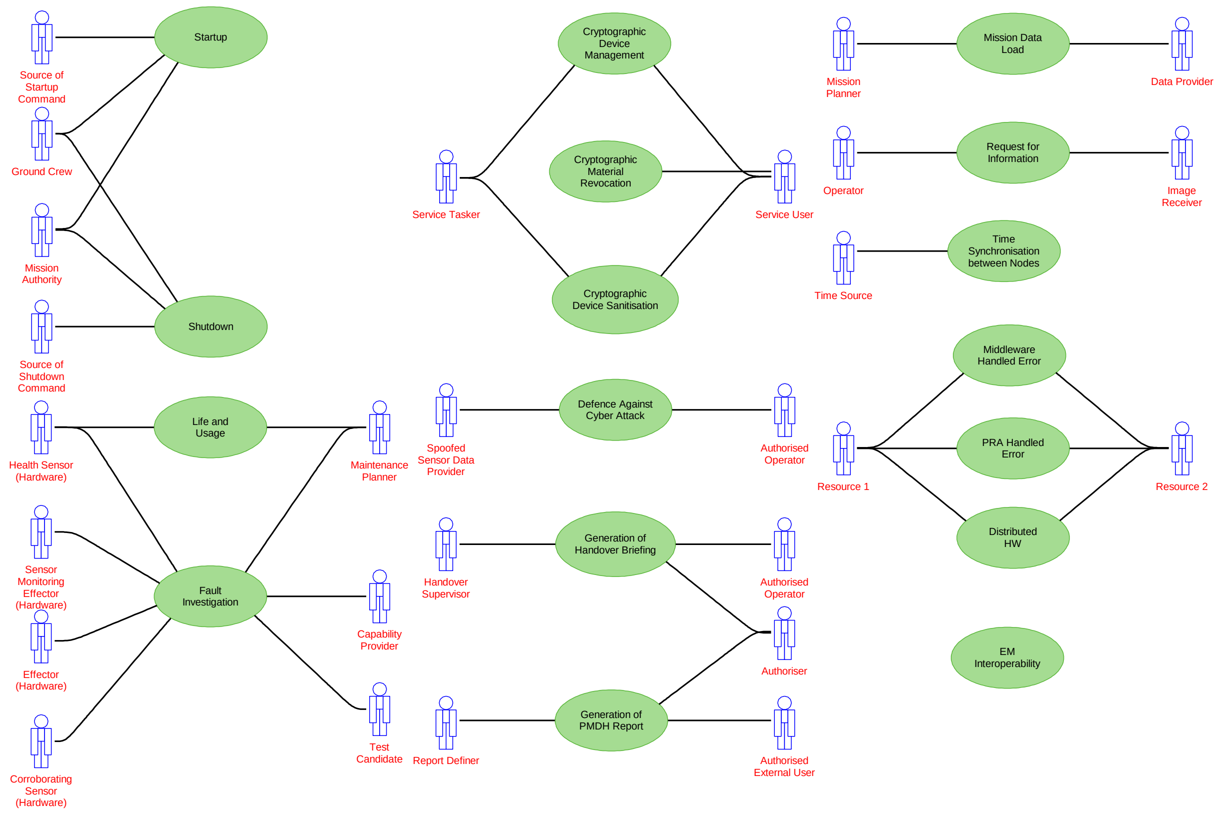


Figure 174: Support Functions Views UC Diagram

C.2.6.1 Startup and Shutdown

C.2.6.1.1 Startup

This IV illustrates the coordination of startup via Asset Transitions, with support from Interlocks. The startup process includes activating the internal power, pumping fuel to the engines and triggering the engine ignition once appropriate authorisation has been obtained.

This IV assumes:

- The Exploiting Platform is starting up to an operational mode.
- The Exploiting Platform has been provided with limited (potentially external) power as required to power up and allow built-in tests to be run.
- The Exploiting Platform has been provided with HMI and/or communications to allow Authorisation to confirm required permissions for startup.
- The Exploiting Platform startup can be completed normally.

This IV excludes:

- Initial power up (possibly by the application of external power) to bring the system to the level of operation required to control safe startup.
- Any required Network Initialisation to support communications as part of startup.
- Required communication support between the Exploiting Platform and external actors.
- Internal capability assessment by a component.
- Pre-mission data load and the loading of physical stores.
- Carrying out post-startup role fit discovery.

C.2.6.1.1.1 Pre-Conditions

- The vehicle is in a configuration ready for transfer to internal power and engine start.
- The vehicle internal power and propulsion are inactive.

C.2.6.1.1.2 View

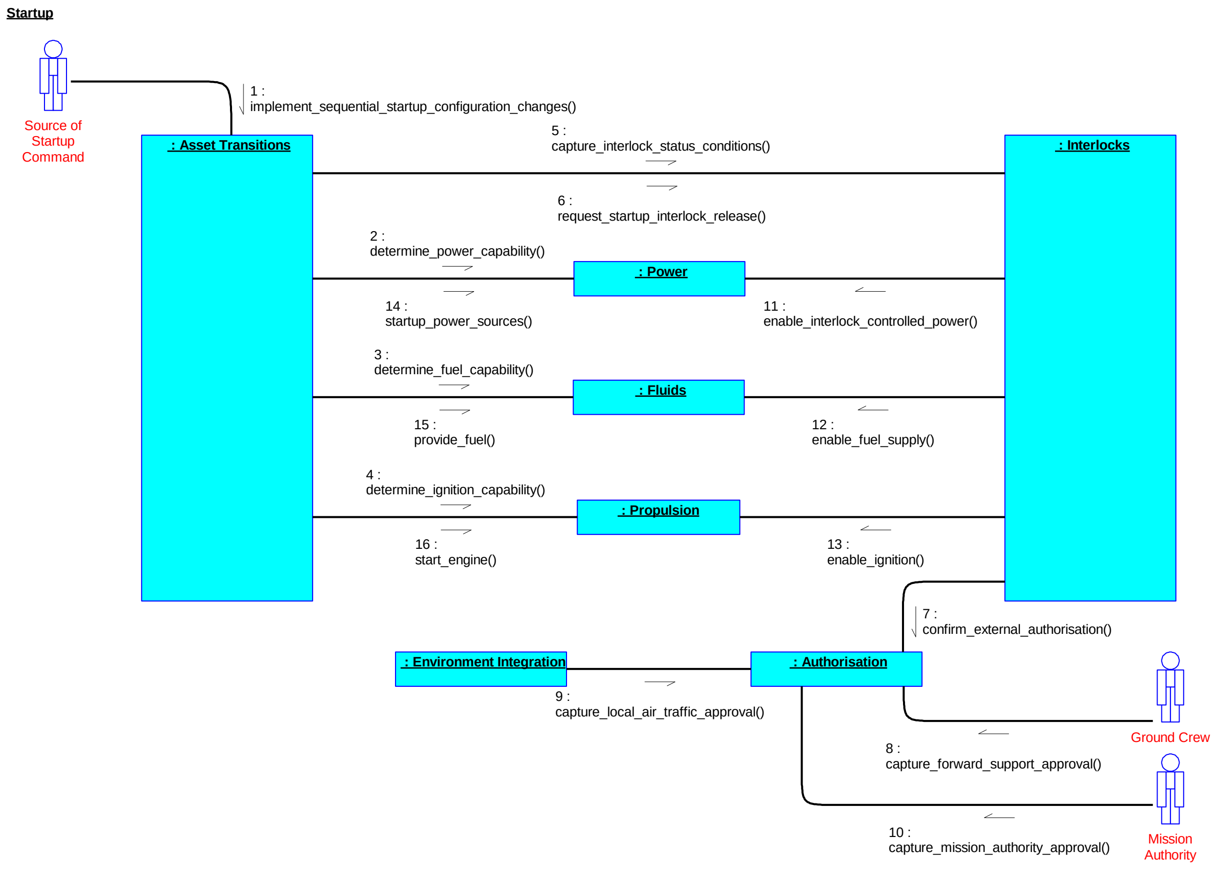

Startup
Source of Startup Command
1 : implement_sequential_startup_configuration_changes()
: Asset Transitions
: Interlocks
5 : capture_interlock_status_conditions()
6 : request_startup_interlock_release()
2 : determine_power_capability()
: Power
14 : startup_power_sources()
11 : enable_interlock_controlled_power()
3 : determine_fuel_capability()
: Fluids
15 : provide_fuel()
12 : enable_fuel_supply()
4 : determine_ignition_capability()
: Propulsion
16 : start_engine()
13 : enable_ignition()
7 : confirm_external_authorisation()
: Environment Integration
: Authorisation
9 : capture_local_air_traffic_approval()
Ground Crew
8 : capture_forward_support_approval()
Mission Authority
10 : capture_mission_authority_approval()

Figure 175: Startup IV

The Source of Startup Command sends an instruction to carry out vehicle startup to Asset Transitions.

Power, Fluids and Propulsion report their capability to Asset Transitions in accordance with the Capability Management PYRAMID concept.

Interlock status conditions are captured by Interlocks, along with conditions (such as external authorisation) required to ensure that the vehicle can safely startup.

Authorisation confirms approval from the Ground Crew attending the vehicle (such as safe distances to prevent hazards from radiating communications), local traffic approval (if not provided directly to a Mission Authority) via Environment Integration and that final approval from the Mission Authority has been received for each stage of startup.

Interlocks confirms that the required interlock status conditions have been met enabling Power, Fluids and Propulsion to move to their active states.

During this process, the vehicle may be required to provide a ground crew communications relay between Ground Crew and the Mission Authority in accordance with the Human Communications IV.

Asset Transitions implements the startup configuration changes. Power and Fluids move to their active states then the engine is started via the Propulsion.

C.2.6.1.1.3 Post-Conditions

- The vehicle power and engine are now in an active state.
- The Power component is able to apply power to equipment as required.

C.2.6.1.1.4 Actors

Ground Crew

The ground crew (and attendant ground support equipment) responsible for the vehicle, and who may be required to give approval for the startup or shutdown of vehicle (either that the vehicle is in a safe state or that the ground crew is safely clear of the vehicle prior to startup).

Mission Authority

The external authority responsible for authorising the vehicle startup or shutdown.

Source of Startup Command

The source of the instruction to Asset Transitions to implement a startup configuration. This may represent a direct instruction from an external entity or an instruction from a component higher in the control architecture.

C.2.6.1.2 Shutdown

This IV illustrates the coordination of a shutdown via Asset Transitions, with support from Interlocks. The shutdown process confirms that shutdown is authorised and then deactivates the Exploiting Platform's engine and power supply.

This IV assumes:

- The Exploiting Platform's engine and internal power is active, though the engine is at an idle state.
- The Exploiting Platform has reached its final position and taxiing activities are completed.
- External power is available if required to support shutdown of internal power.
- The Exploiting Platform shutdown can be completed normally.

This IV excludes:

- Post-mission data extraction and the removal of physical stores.
- Internal capability assessment by a component in accordance with the Capability Management PYRAMID concept.
- Required communication support between the Exploiting Platform and external actors.
- Sanitisation of the Exploiting Platform (for example by Cryptographic Materials and Storage) to make the Exploiting Platform secure from a data perspective.

C.2.6.1.2.1 Pre-Conditions

- The vehicle is in a configuration ready for shutdown.

C.2.6.1.2.2 View

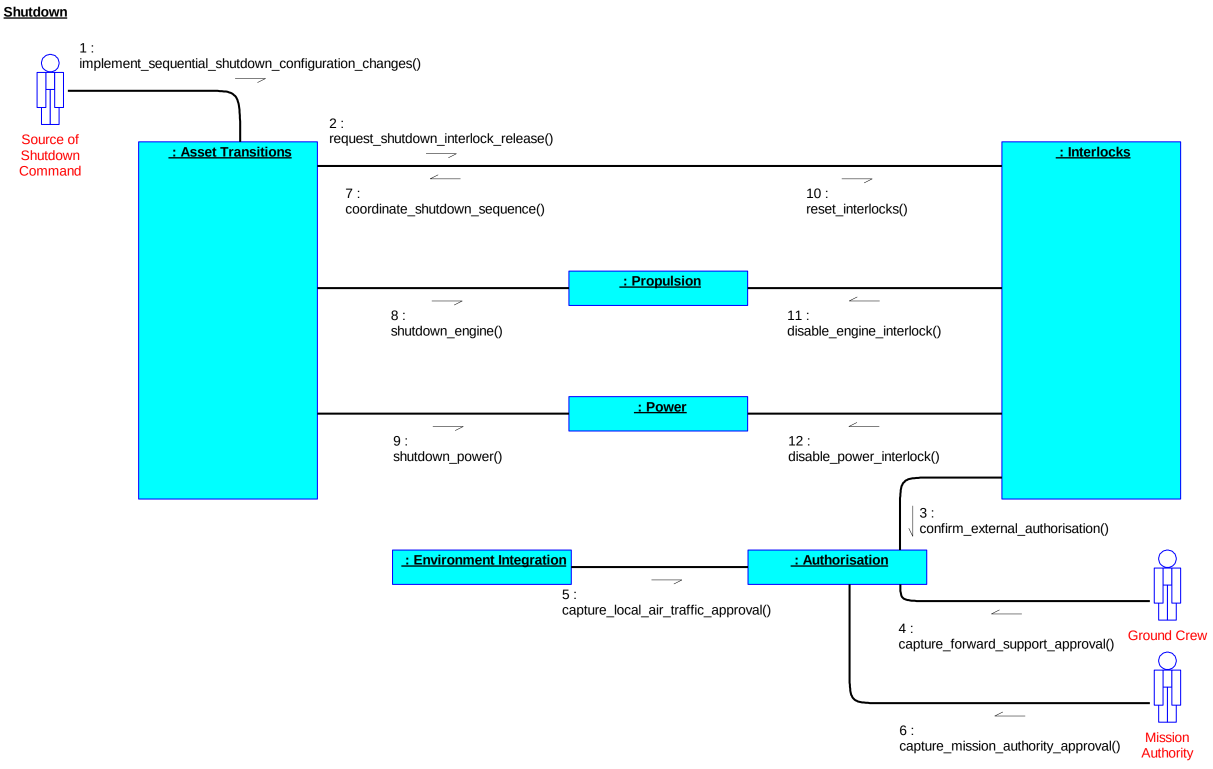

Shutdown
Source of Shutdown Command
1 : implement_sequential_shutdown_configuration_changes()
: Asset Transitions
2 : request_shutdown_interlock_release()
: Interlocks
7 : coordinate_shutdown_sequence()
10 : reset_interlocks()
: Propulsion
8 : shutdown_engine()
11 : disable_engine_interlock()
: Power
9 : shutdown_power()
12 : disable_power_interlock()
3 : confirm_external_authorisation()
: Environment Integration
: Authorisation
5 : capture_local_air_traffic_approval()
4 : capture_forward_support_approval()
Ground Crew
6 : capture_mission_authority_approval()
Mission Authority

Figure 176: Shutdown IV

The Source of Shutdown Command sends an instruction to carry out vehicle shutdown to Asset Transitions.

Asset Transitions confirms through Interlocks that a shutdown is allowed.

Interlocks performs any conditional checks and requests shutdown approval through Authorisation to ensure that the vehicle can safely shutdown.

Authorisation confirms that shutdown approval has been obtained from the required sources:

- Ground Crew attending the vehicle.
- Local traffic approval (if not provided directly to a Mission Authority).
- Final approval has been received from the Mission Authority (as relevant) for each stage of shutdown.

During this process, the vehicle may be required to provide a ground crew communications relay between Ground Crew and the Mission Authority in accordance with the Human Communications IV.

Asset Transitions commands the shutdown configuration changes. Interlocks restores the interlock states to prevent a restart without proper approval and then Power and Propulsion move to their inactive states.

C.2.6.1.2.3 Post-Conditions

- All power has been removed from the aircraft.

C.2.6.1.2.4 Actors

Ground Crew

The ground crew (and attendant ground support equipment) responsible for the vehicle, and who may be required to give approval for the startup or shutdown of vehicle (either that the vehicle is in a safe state or that the ground crew is safely clear of the vehicle prior to startup).

Mission Authority

The external authority responsible for authorising the vehicle startup or shutdown.

Source of Shutdown Command

The source of the instruction to Asset Transitions to implement a shutdown configuration. This may represent a direct instruction from an external entity or an instruction from a component higher in the control architecture.

C.2.6.2 Health Management

C.2.6.2.1 Fault Investigation

This IV demonstrates how a hardware fault is identified by using an anomaly as a trigger event. An anomaly is detected and investigated, resulting in a fault being identified, characterised, and published. This particular IV describes a health change associated with an effector, which is detected by a strategically positioned sensor.

In this scenario the Sensor Monitoring Effector (Hardware) is depicted as being a separate entity in its own right. In some instances the Sensor Monitoring Effector (Hardware) and the Effector (Hardware) may be embodied in the same item of equipment. Any item of equipment that provides information which is relevant to health can be regarded as a health sensor.

It is assumed that:

- The system has no existing faults.

The following considerations are excluded:

- This IV does not demonstrate Health Assessment components working and communicating with other Health Assessment components in a hierarchy, which would be relevant in more complex health scenarios.
- HMI interactions between Health Assessment and the Maintenance Planner.
- Behaviour associated with how Cyber Defence analyses an anomaly.
- Behaviour associated with Test.
- Behaviour associated with how system capability is determined based on a published anomaly or health assessment.
- Problem resolution is excluded from this IV. It does not cover any of the following:
    - Replacing/repairing the fault candidate.
    - Fault accommodation at a hardware level.
    - Replanning the capability.

C.2.6.2.1.1 Pre-Conditions

- No pre-conditions are identified.

C.2.6.2.1.2 View

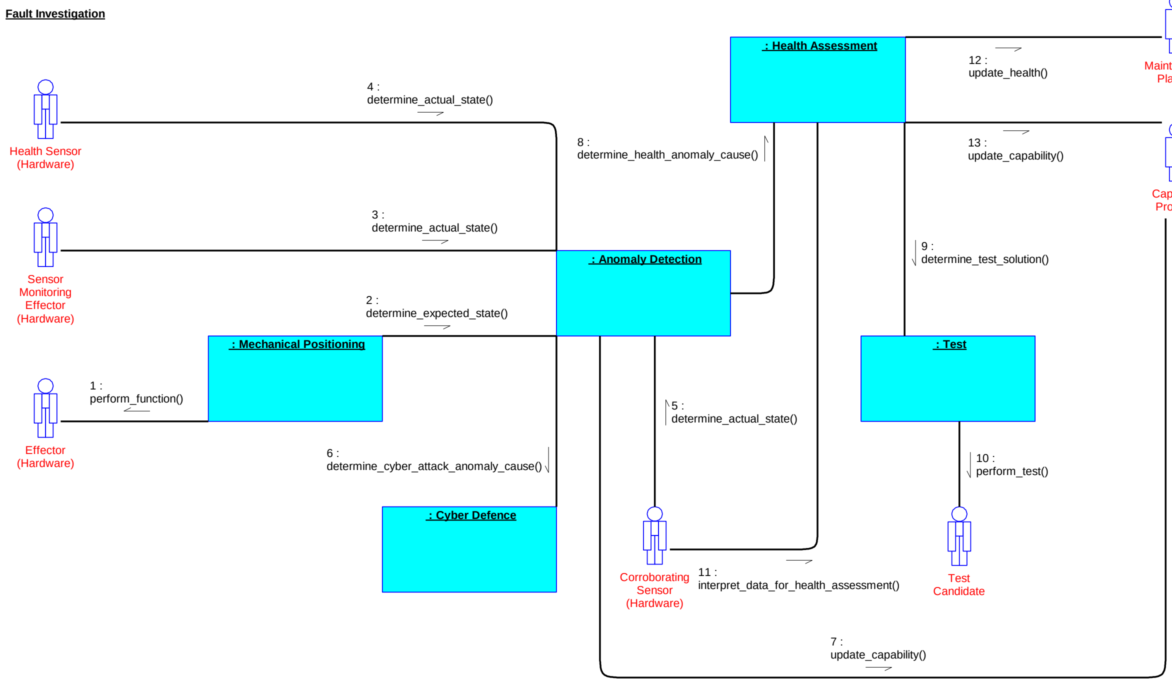

Fault Investigation
: Health Assessment
12 : update_health()
Maintenance Planner
4 : determine_actual_state()
Health Sensor (Hardware)
8 : determine_health_anomaly_cause()
13 : update_capability()
Capability Provider
3 : determine_actual_state()
Sensor Monitoring Effector (Hardware)
9 : determine_test_solution()
: Anomaly Detection
2 : determine_expected_state()
: Mechanical Positioning
: Test
1 : perform_function()
Effector (Hardware)
5 : determine_actual_state()
6 : determine_cyber_attack_anomaly_cause()
10 : perform_test()
: Cyber Defence
Corroborating Sensor (Hardware)
11 : interpret_data_for_health_assessment()
Test Candidate
7 : update_capability()

Figure 177: Fault Investigation IV

Anomaly Detection monitors interactions between PYRAMID components in order to identify interactions that would result in a change of state of the system. If Anomaly Detection identifies an interaction that would result in a change, it will create an expected state (the state the system is expected to transition into when the command is fulfilled).

When the Effector (Hardware) performs a function, Anomaly Detection will determine if the Effector (Hardware) has transitioned into the expected state. It does so by utilising any relevant monitoring sensors to determine the Effector (Hardware) actual state. This can include monitoring any relevant Health Sensor (Hardware), Corroborating Sensor (Hardware) and Sensor Monitoring Effector (Hardware). Once the Effector (Hardware) actual state has been determined it is cross referenced against the expected state that was created. If the actual state and expected state do not concur, this is classified as an observed anomaly.

Anomaly Detection publishes the observed anomaly to Cyber Defence, any relevant Capability Providers and Health Assessment.

Health Assessment analyses the observed anomaly in order to determine if a health issue is the root cause of the anomaly. If the root cause cannot be ascertained simply by analysing the observed anomaly, Health Assessment can attempt to gather further information. This can be achieved by looking at information provided by Corroborating Sensor (Hardware), or by requesting Test to initiate any relevant built in tests. Initiating a test is covered in more detail in the Test PYRAMID concept. Any information provided by the test becomes available to Health Assessment and Anomaly Detection via a Corroborating Sensor (Hardware).

Health Assessment analyses the additional information to attempt to determine the root cause of the anomaly. It may continue to request further information to refine its assessment and increase the certainty of its final answer.

Once Health Assessment has determined the root cause of the anomaly, it will notify the Maintenance Planner so that a maintenance solution can be identified and also notify the Capability Provider so that it can assess how the failure affects its capabilities.

C.2.6.2.1.3 Post-Conditions

- The Maintenance Planner and relevant Capability Providers are aware of health changes.

C.2.6.2.1.4 Actors

Capability Provider

A PYRAMID component that provides one or more capabilities to the overall system. This capability may be affected by one or more health changes.

Corroborating Sensor (Hardware)

A sensor that provides additional information to aid health assessment in its root cause analysis (e.g. test results).

Effector (Hardware)

An item of equipment that is responsible for performing a function.

Health Sensor (Hardware)

A sensor that generates health data.

Maintenance Planner

The person responsible for analysing the health data and deciding on the most appropriate maintenance action.

Sensor Monitoring Effector (Hardware)

A sensor that is strategically positioned to monitor a specific property of an effector.

Test Candidate

The item of equipment responsible for performing the test.

C.2.6.2.2 Life and Usage

This IV demonstrates how life and usage information is collected, analysed and reported.

This scenario considers a health specific sensor whose role is purely to collect health data about a discrete element of a system so that Health Assessment can calculate total amount of usage consumed by that discrete element, calculate total remaining usage and then identify if this information needs reporting. Life monitoring uses a similar mechanism to monitor the duration an element is in operation.

It is assumed that:

- The system has no existing life and usage exceedances.

The following considerations are excluded:

- Behaviour associated with abnormal life or usage consumption. This would be treated as an anomalous event and be handled in the normal way by Anomaly Detection.
- HMI interactions between Health Assessment and the Maintenance Planner.

C.2.6.2.2.1 Pre-Conditions

- Life and usage thresholds are pre-defined and supplied to Health Assessment prior to initialisation.

C.2.6.2.2.2 View

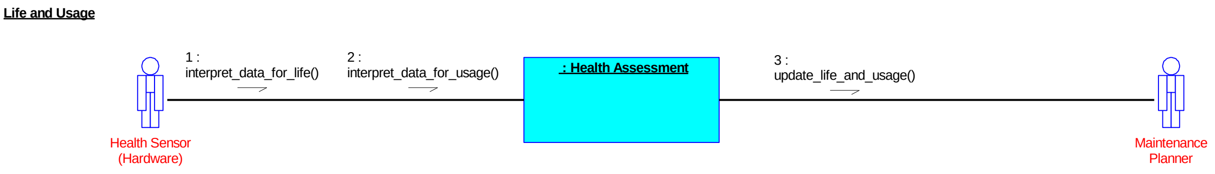

Life and Usage
1 : interpret_data_for_life()
2 : interpret_data_for_usage()
: Health Assessment
3 : update_life_and_usage()
Health Sensor (Hardware)
Maintenance Planner

Figure 178: Life and Usage IV

Health Sensor (Hardware) are strategically placed in order to collect health information on certain aspects of discrete elements within a system. A Health Sensor (Hardware) provides health data to Health Assessment. Health Assessment analyses the health data to determine the total amount of life or usage consumed by a discrete element and then determines the total amount of remaining life or usage.

Health Assessment compares the total remaining life or usage against a predefined threshold and determines that the threshold has been exceeded.

Health Assessment provides the life and usage information to the Maintenance Planner so that they can update their life and usage records and so that a maintenance solution can be identified.

C.2.6.2.2.3 Post-Conditions

- The Maintenance Planner is aware that the usage threshold has been exceeded.

C.2.6.2.2.4 Actors

Health Sensor (Hardware)

A sensor that generates health data.

Maintenance Planner

The person responsible for analysing the health data and deciding on the most appropriate maintenance action.

C.2.6.3 EM Interoperability

The system has to manage electromagnetic emissions in order to achieve EM interoperability. This IV illustrates how the EM emissions are managed to enable interoperability by using the blanking interoperability mechanism.

Even though this use case only considers blanking and sensor processing, there are other interoperability mechanisms that could have been used in its place. Example EM interoperability mechanisms are:

- Blanking - Informing a receiver to not receive anything.
- Rejection - Reject any data received.
- Suppression - Denial of transmission.
- Processing - Eliminating spurious data by post-processing.

This use case only uses Effectors and Sensors. Other types of EM resources (e.g. Communicator) can be managed in the same manner at the same time.

The numerical bandwidths used in this use case are fictitious and do not represent the bandwidths at which these types of resource would function.

It is assumed that:

- Use of the managed resources has been planned in adherence with the Resource Management PYRAMID concept.

The following considerations are excluded:

- Influencing the direction or zones of transmission and reception associated with EM interoperability. I.e. the equipment is assumed to spatially and chronologically interfere, and methods such as repositioning the transmitters or receivers, beamforming, or rescheduling other activities (in accordance with the Resource Management PYRAMID concept) are not possible in this IV.
- The mechanisms used for detecting and reporting threats to Countermeasures.

C.2.6.3.1 Pre-Conditions

- The conditions for the use of EM transmissions have been loaded into Operational Rules and Limits.
- Spectrum has been loaded with the EM interoperability mechanisms.
- No countermeasures are active.

C.2.6.3.2 View

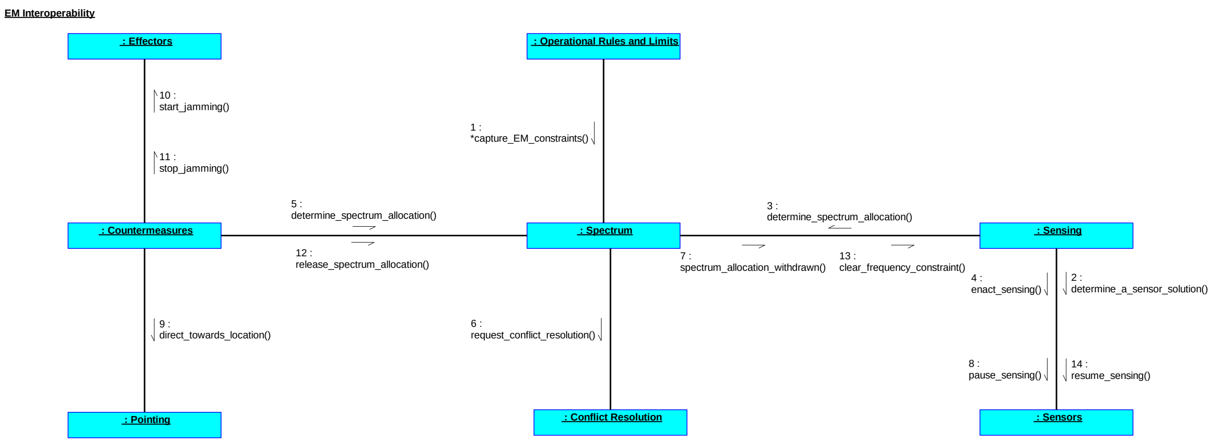

EM Interoperability
: Effectors
: Operational Rules and Limits
10 : start_jamming()
11 : stop_jamming()
1 : *capture_EM_constraints()
5 : determine_spectrum_allocation()
12 : release_spectrum_allocation()
3 : determine_spectrum_allocation()
: Countermeasures
: Spectrum
: Sensing
7 : spectrum_allocation_withdrawn()
13 : clear_frequency_constraint()
4 : enact_sensing()
2 : determine_a_sensor_solution()
9 : direct_towards_location()
6 : request_conflict_resolution()
8 : pause_sensing()
14 : resume_sensing()
: Pointing
: Conflict Resolution
: Sensors

Figure 179: EM Interoperability IV

Operational Rules and Limits provides limits to Spectrum that affect spectrum usage. Sensing provides a requirement, including to use the 2 GHz frequency, to Sensors. Sensors determines a solution that satisfies the requirement. Sensing requests the allocation of 2 GHz frequency from Spectrum. Once allocated, Sensing informs Sensors it can enact the solution and perform the sensing activity.

During the operation, Countermeasures is made aware of a threat and requests a spectrum allocation of 2 GHz frequency from Spectrum.

Spectrum identifies a conflict and requests Conflict Resolution to facilitate resolving it.

Conflict Resolution arbitrates between the demands from Countermeasures and the existing Sensors allocation. The request from the Countermeasures component is a high priority demand which incurs an immediate arbitration decision conforming to the rules and strategies configured within Conflict Resolution. Spectrum withdraws the allocation of 2 GHz frequency from Sensing and Sensing informs Sensors to pause the sensing activity.

The 2 GHz frequency is allocated to Countermeasures as requested. Countermeasures instructs Pointing to direct the effects of the jamming pod towards the threat location, and instructs Effectors to activate jamming.

Once the threat is avoided, Countermeasures instructs Effectors to de-activate jamming, and informs Spectrum to release the allocation.

Spectrum removes the blanking restriction and re-allocates the use of the 2 GHz frequency to the sensing activity. Once re-allocated, Sensing directs Sensors to continue performing the sensing activity.

C.2.6.3.3 Post-Conditions

- Countermeasure emissions have completed.
- Sensor equipment is operating at 2 GHz frequency.

C.2.6.4 Cryptographic Management

C.2.6.4.1 Cryptographic Device Management

This IV illustrates the configuration of encryption in response to a request for integrity protection for data-in-transit. Other permutations of confidentiality, integrity and availability protection configuration for other uses follow the same pattern.

It is assumed that:

- Cross domain behaviour will not change if multiple security domains are involved.
- If cryptographic material is distributed encrypted (e.g. black keys) then the appropriate cryptographic material for decryption has already been distributed. Note: at some point in the hierarchy of keys, unencrypted cryptographic material has to be supplied (e.g. red keys) otherwise encrypted material can never be decrypted.
- The cryptographic device is not pre-loaded with cryptographic materials.

The following considerations are excluded:

- The flow of data requiring cryptographic transformation is not shown.

C.2.6.4.1.1 Pre-Conditions

- The plan of which cryptographic devices are to be used for which task in the mission is already loaded in the system.
- Cryptographic material is already loaded in the system in an appropriate cryptographic store.
- The data exchange requirements are pre-configured within the relevant components.

C.2.6.4.1.2 View

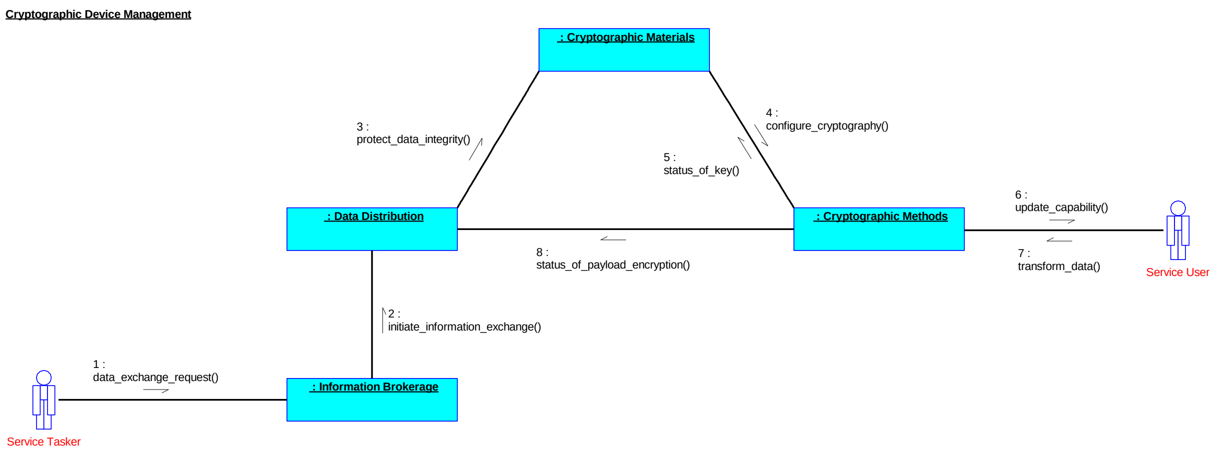

Cryptographic Device Management
: Cryptographic Materials
3 : protect_data_integrity()
4 : configure_cryptography()
5 : status_of_key()
: Data Distribution
: Cryptographic Methods
6 : update_capability()
7 : transform_data()
Service User
8 : status_of_payload_encryption()
2 : initiate_information_exchange()
1 : data_exchange_request()
: Information Brokerage
Service Tasker

Figure 180: Cryptographic Device Management IV

Following a request for an information exchange from the Service Tasker, Information Brokerage determines that integrity protection is required. This requirement to carry out a secure information exchange is placed onto Data Distribution and Data Distribution requests a protection solution from the Cryptographic Materials component.

Cryptographic Materials determines what protection is required and provides the required information to Cryptographic Methods, including any cryptographic key material needed by the Cryptographic Methods component. Cryptographic Methods reports security related information (status of keys) back to Cryptographic Materials and also provides evidence to the Service User of a change in its capability to provide cryptographic transformations.

The Service User can now make use of the required Cryptographic Methods capabilities, which Cryptographic Methods reports the progress of (e.g. status of payload protection) to Data Distribution.

C.2.6.4.1.3 Post-Conditions

- The appropriate encryption has been prepared to preserve the integrity of data-in-transit.

C.2.6.4.1.4 Actors

Service Tasker

The user or internal component that requests a service that indirectly requires cryptographic support.

Service User

The user of the cryptography service.

C.2.6.4.2 Cryptographic Material Revocation

This IV illustrates the revocation of a cryptographic certificate that has been identified as compromised, and is currently used in data-at-rest protection. Multiple instances of Cryptographic Methods are involved in the deployment, and are using shared cryptographic material. Other permutations of revocation will follow the same pattern.

It is assumed that:

- Cross-domain behaviour will not change if multiple security domains are involved.

The following considerations are excluded:

- The use of cryptography is not included in this interaction diagram only the revocation of cryptographic material.
- Actions triggered by Storage as a result of the certificate being compromised, which do not relate to cryptography are not shown.

C.2.6.4.2.1 Pre-Conditions

- The plan of which cryptographic devices are to be used for which activities in the mission are already loaded in the system.
- Cryptographic material is already loaded in crypto devices.

C.2.6.4.2.2 View

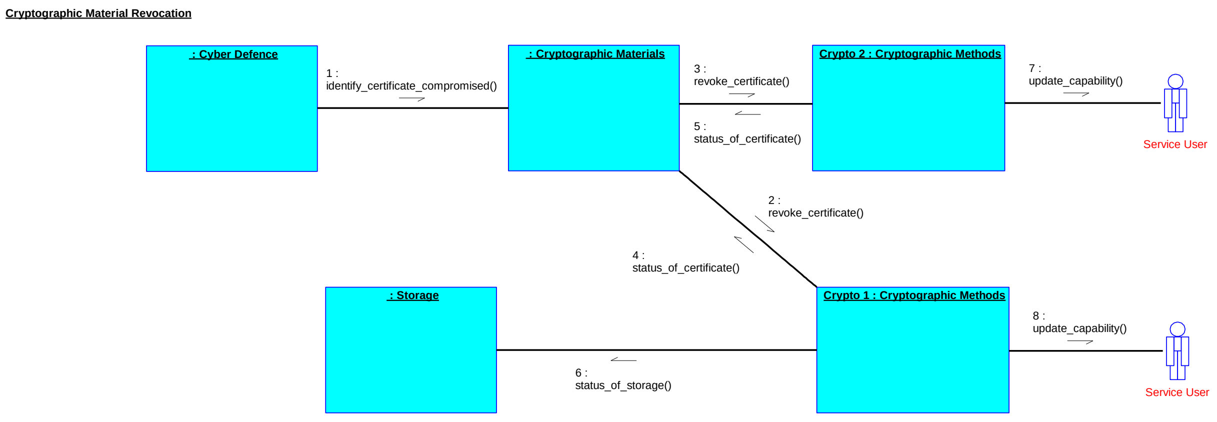

Cryptographic Material Revocation
: Cyber Defence
1 : identify_certificate_compromised()
: Cryptographic Materials
3 : revoke_certificate()
5 : status_of_certificate()
Crypto 2 : Cryptographic Methods
7 : update_capability()
Service User
2 : revoke_certificate()
4 : status_of_certificate()
: Storage
6 : status_of_storage()
Crypto 1 : Cryptographic Methods
8 : update_capability()
Service User

Figure 181: Cryptographic Material Revocation IV

The Cyber Defence component, in consultation with other components (not shown) determines the impact on the system of a reduced trust in a storage resource. Cyber Defence informs Cryptographic Materials that cryptographic material has been compromised. Cryptographic Materials then determines which certificate has been compromised.

As the certificate has been compromised the certificate must not be used by any device. The Cryptographic Materials component is responsible for keeping track of keys, algorithms and certificates in the system, so will determine which devices are using the certificate and request certificate revocation from all relevant Cryptographic Methods instances. As multiple Cryptographic Methods instances are to be updated Cryptographic Materials will determine the best order to perform this request.

Cryptographic Methods instances will inform the Cryptographic Materials component that the certificate has been revoked (as this component is responsible for security related information) and will inform Storage of the status of the compromised store (i.e. protection service no longer available for this store). Cryptographic Methods instances also provide evidence to the Service Users of a change in their capability to provide cryptographic transformations.

C.2.6.4.2.3 Post-Conditions

- Cryptographic material has been successfully revoked from multiple devices.

C.2.6.4.2.4 Actors

Service User

The user of the cryptography service.

C.2.6.4.3 Cryptographic Device Sanitisation

This IV illustrates the sanitisation of a communication cryptographic device as it is no longer required for the mission. Other permutations of sanitisation, including emergency sanitisation follow a similar pattern.

It is assumed that:

- Only the single device needs sanitising, if other devices needed sanitising this would be managed by the Cryptographic Materials component.

The following considerations are excluded:

- Non-cryptographic actions not shown in this IV, for examples of this behaviour see the Communications Views.

C.2.6.4.3.1 Pre-Conditions

- The plan of which cryptographic devices are to be used for which activities in the mission are already loaded in the system.
- Cryptographic material is already loaded in cryptographic devices.

C.2.6.4.3.2 View

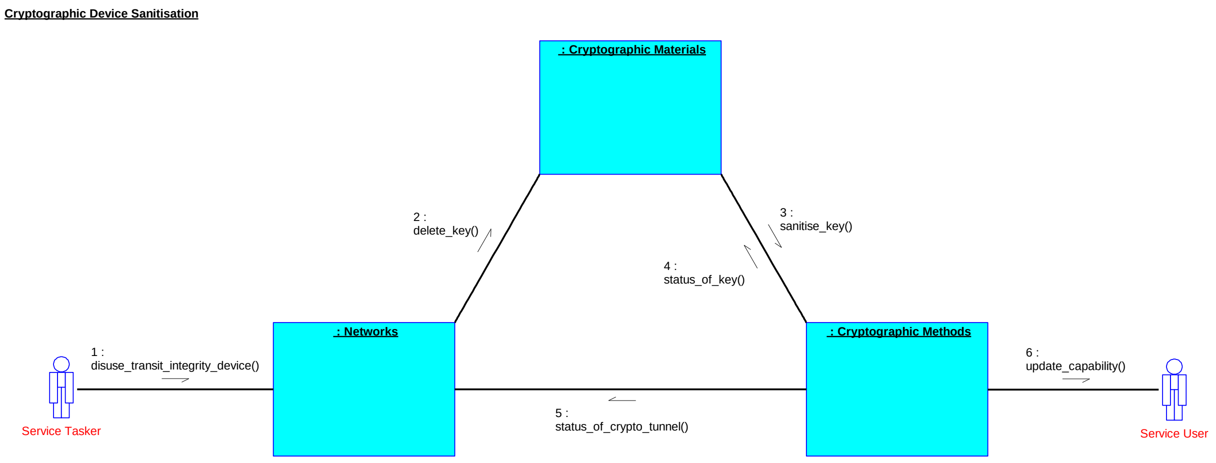


Figure 182: Cryptographic Device Sanitisation IV

The Service Tasker will inform the Networks component that certain parts of the communications infrastructure are no longer required for the mission. Networks informs Cryptographic Materials that as part of the changes a crypto device is no longer needed. Cryptographic Materials determines that the redundant device can be cleared.

The Cryptographic Materials component determines which cryptographic material can be removed and requests sanitisation of the related Cryptographic Methods component.

On completion of the sanitisation Cryptographic Methods will inform the Cryptographic Materials component, e.g. provide status of cryptography key(s). Cryptographic Materials will mark the key material as no longer being used on that device.

Cryptographic Methods will also inform the Networks component and the Service User that the encryption service it was providing is no longer available. Based on this information the Networks component will determine that the encrypted tunnel is no longer able to pass traffic, and the Service User will no longer make use of the affected cryptographic transformation service, which was the expected outcome.

C.2.6.4.3.3 Post-Conditions

- Cryptographic material has been successfully sanitised.

C.2.6.4.3.4 Actors

Service Tasker

The user or internal component that requests a service that indirectly requires cryptographic support.

Service User

The user of the cryptography service.

C.2.6.5 Defence Against Cyber Attack

This IV covers a cyber attack scenario whereby a vulnerability has been exploited by an adversary and a malicious payload activated.

Detection of an exploitation is achieved within the PRA by a multi-component analysis of multiple event sources that together can be used to identify a problem (e.g. a process gaining extra privileges combined with unusually large amounts of data being accessed). The "effect" is detected by monitoring the capability and health of the system and identifying any reduced or unexpected performance, analysing the cause and reacting appropriately.

This IV depicts a single cyber attack which contributes to a larger cyber attack. Here, an adversary has successfully delivered a malicious payload which manipulates sensor data resulting in spoofed tracks being generated. In response Anomaly Detection, Cyber Defence and Health Assessment work together to identify and, if possible, mitigate the attack.

This section follows the Health Management, Control Architecture and Cyber Defence PYRAMID concepts. On the IV the Anomaly Detection component is a single instance associated with Data Fusion. If multiple components were displayed in Figure 183: Defence Against Cyber Attack IV there would be multiple instances, which is aligned to the Health Management PYRAMID concept.

This IV only considers spoofed track(s) as an example. If other components have been attacked or other types of cyber attack methods have been deployed, the general interactions between the relevant components wouldn’t change.

It is assumed that:

- Defence against cyber attack will be fulfilled using a "Defence in Depth" approach, comprising personnel, physical, procedural and technical controls against cyber threats.

The following considerations are excluded:

- How an adversary successfully delivered the malicious payload.
- Personnel, physical and procedural controls are outside the scope of the PRA, as are technical controls residing below the application layer of the computing stack.
- Interactions between HMI components are not shown as these are shown elsewhere.
- Generation of a security log.

C.2.6.5.1 Pre-Conditions

- The cyber attack contingency tactics have been loaded to Tasks.
- Cyber Defence is loaded with and has knowledge of cyber attacks and their methods, including those associated with data fusion.
- An adversary has successfully delivered the malicious payload that manipulates sensor data to the air vehicle.

C.2.6.5.2 View

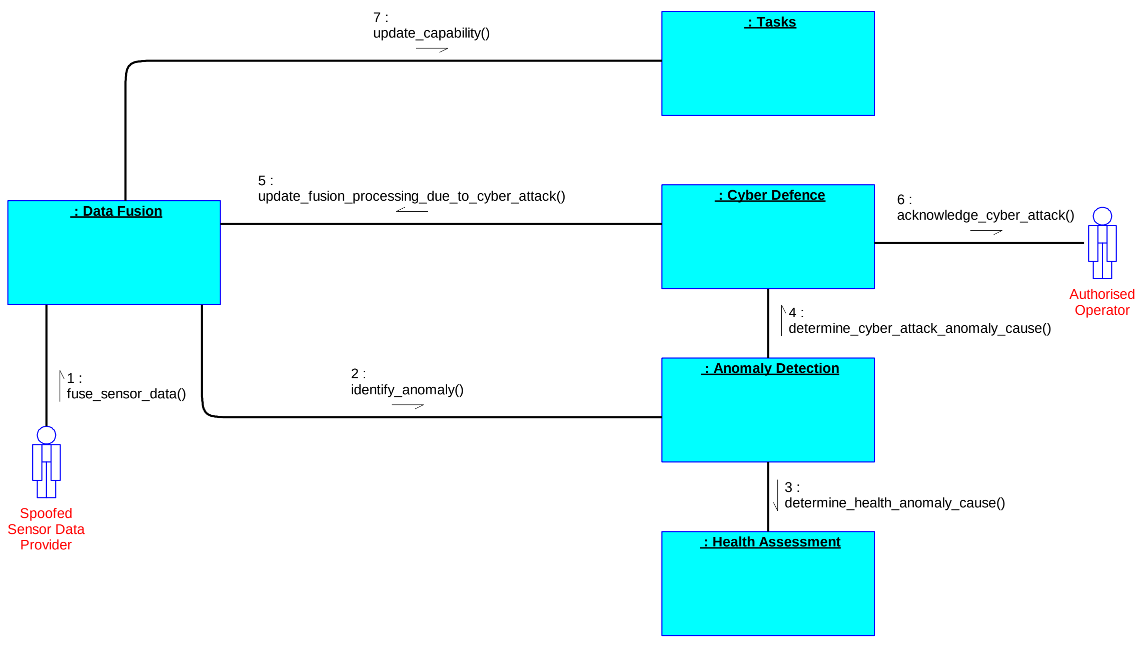

Defence Against Cyber Attack
7 : update_capability()
: Tasks
5 : update_fusion_processing_due_to_cyber_attack()
: Data Fusion
: Cyber Defence
6 : acknowledge_cyber_attack()
Authorised Operator
4 : determine_cyber_attack_anomaly_cause()
1 : fuse_sensor_data()
2 : identify_anomaly()
: Anomaly Detection
Spoofed Sensor Data Provider
3 : determine_health_anomaly_cause()
: Health Assessment

Figure 183: Defence Against Cyber Attack IV

The Spoofed Sensor Data Provider provides manipulated sensor data to Data Fusion which then uses that data to maintain the interpretations.

The interpretations and other health related data are then collected by Anomaly Detection so that it can be interpreted and the anomalies can be identified.

Anomaly Detection identifies that there are anomalous Data Fusion results based on interpreting current results against a baseline of previous results. Anomaly Detection provides the anomalies to Data Fusion, Cyber Defence and Health Assessment. Cyber Defence and Health Assessment simultaneously assess the anomaly. Health Assessment does not identify a hardware issue and Cyber Defence determines that a cyber attack has taken place, in this case by comparing the results against a knowledge base of known cyber attack types, malware and threats.

Cyber Defence identifies the possible actions in response to the cyber attack which results in informing Data Fusion to update its processing capability. Additionally, where a pre-determined notification threshold has been exceeded, the Authorised Operator is informed that a cyber attack has been successful. Data Fusion determines that any sensor data received from the Spoofed Sensor Data Provider should be ignored and informs Tasks of a change in its capability.

Tasks then produces a mitigation scheme.

C.2.6.5.3 Post-Conditions

- Tasks has determined a mitigation scheme.

C.2.6.5.4 Actors

Authorised Operator

Any person with validated credentials allowed to interact with the system to carry out a system role.

Spoofed Sensor Data Provider

A software component of the system, such as Sensor Products, which has been corrupted into providing spoofed sensor data.

C.2.6.6 Generation of Reports

C.2.6.6.1 Generation of Handover Briefing

This IV shows how a handover briefing is created to support the handover from one Authorised Operator to another. The briefing provides the incoming Authorised Operator with a summary of the mission status at the time of the handover. The briefing will follow a general format, but the precise details will depend on the mission.

It is assumed that:

- All required data has been successfully recorded.

The following considerations are excluded:

- The HMI and communication components required for the interface with the operators.

C.2.6.6.1.1 Pre-Conditions

None.

C.2.6.6.1.2 View

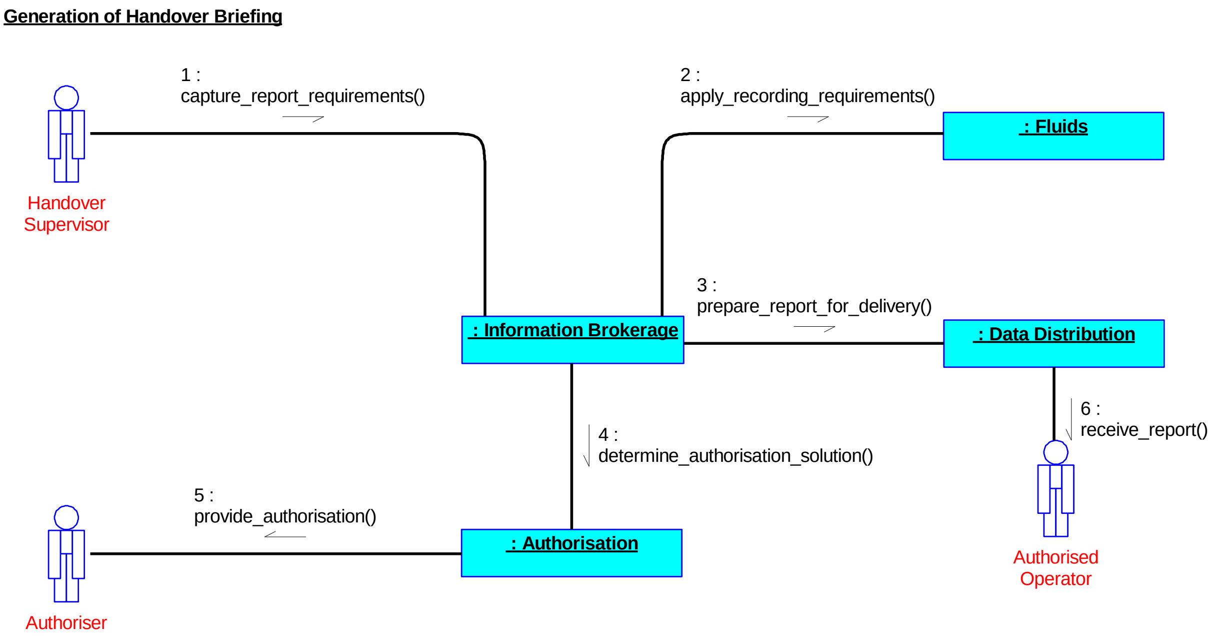

Generation of Handover Briefing
1 : capture_report_requirements()
2 : apply_recording_requirements()
: Fluids
Handover Supervisor
3 : prepare_report_for_delivery()
: Information Brokerage
: Data Distribution
6 : receive_report()
4 : determine_authorisation_solution()
5 : provide_authorisation()
: Authorisation
Authorised Operator
Authoriser

Figure 184: Generation of Handover Briefing IV

The Handover Supervisor defines the required handover briefing. The precise contents of the briefing will depend on the mission details, so the briefing is defined in terms of rules for the data items that are to be included, based on events and mission conditions. Information Brokerage captures these requirements for the handover briefing and imposes requirements on the relevant components (represented here by Fluids) to record the necessary data.

During the mission, Fluids writes the fuel contents data to storage hardware by using the middleware.

When the time comes for a handover, Information Brokerage applies the briefing rules and, based on mission events and conditions, instructs Data Distribution to collate the fuel contents records and other data items required. Once the report content has been collated, Information Brokerage requests Authorisation to attain release authorisation from the Authoriser.

The Authoriser fulfils the request to authorise release of the briefing to the Authorised Operator.

Information Presentation (not shown) then presents the briefing report.

C.2.6.6.1.3 Post-Conditions

- The handover briefing has been released to the Authorised Operator who is taking over control.

C.2.6.6.1.4 Actors

Authorised Operator

An operator whose control role will be handed over to another operator during the course of the planned mission.

Authoriser

An authorised operator of the system who is able to authorise the release of a handover report, or mission data for PMDH.

Handover Supervisor

An operator who will supervise the handover of control roles between operators during the course of a mission.

C.2.6.6.2 Generation of PMDH Report

This IV shows how a set of mission data is generated for Post Mission Data Handling (PMDH) by an Authorised External User. The data set will include detailed technical information about the conduct of the mission. The data set will follow a general format, but the precise details will depend on the mission.

It is assumed that:

- All required data has been successfully recorded.

The following considerations are excluded:

- The HMI and communication components required for the interface with the operators.

C.2.6.6.2.1 Pre-Conditions

- The mission has completed and all necessary data has been recorded.

C.2.6.6.2.2 View

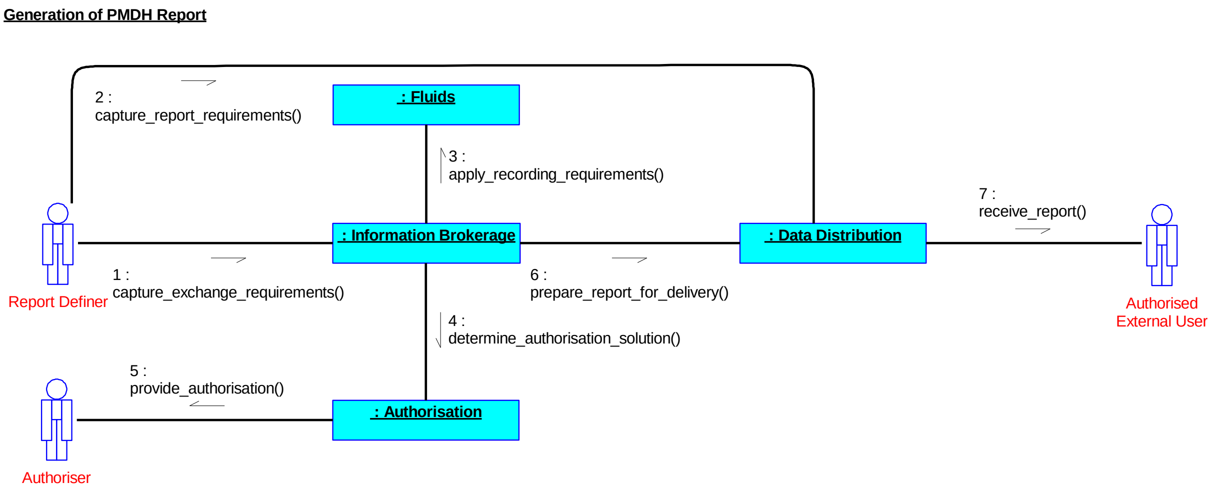

Generation of PMDH Report
2 :
capture_report_requirements()
: Fluids
3 :
apply_recording_requirements()
7 :
receive_report()
Report Definer
: Information Brokerage
: Data Distribution
Authorised External User
1 :
capture_exchange_requirements()
6 :
prepare_report_for_delivery()
4 :
determine_authorisation_solution()
5 :
provide_authorisation()
: Authorisation
Authoriser

Figure 185: Generation of PMDH Report IV

The Report Definer defines the PMDH data set. The precise contents of the data set will depend on the mission details, so it is defined in terms of rules for the data items that are to be included, based on events and mission conditions. Information Brokerage captures the required exchanges to collate the PMDH data set, and Data Distribution captures the requirement to collate the report. Information Brokerage imposes requirements on the relevant components (represented here by Fluids) to record the necessary data.

During the mission, Fluids writes the fuel contents data to storage hardware by using the middleware. During the mission, an error condition arises which causes Fluids to create a software error statement.

After the mission, Information Brokerage identifies the data items required by the PMDH data set rules based on the events and mission conditions. This includes the fuel contents and the software error statement. Information Brokerage requests Authorisation to obtain release authorisation from the Authoriser. The Authoriser fulfils the request for authorisation to release the data set. Data Distribution collates the PMDH data set from the relevant components, which is released to the Authorised External User for analysis.

C.2.6.6.2.3 Post-Conditions

- The PMDH data set has been released to the Authorised External User for analysis.

C.2.6.6.2.4 Actors

Authorised External User

An operator external to the operational system who will receive mission data for analysis once the mission is complete.

Report Definer

An operator of the system who is authorised to define PMDH data sets.

Authoriser

An authorised operator of the system who is able to authorise the release of a handover report, or mission data for PMDH.

C.2.6.7 Mission Data Load

This IV shows how a PYRAMID compliant deployment can derive and assemble the information that an operational system requires (the Mission Data Load (MDL)), to support a mission. The IV does not exclude the possibility that the generation of the MDL could be executed in an independent mission planning system (see Operational Support PYRAMID concept).

It is assumed that:

- All necessary data libraries are available to the mission planning system.

The following considerations are excluded:

- The mechanism for physical transfer of the MDL to the operational system.
- The acquisition of data (from source components) by Data Distribution in preparation for delivery.
- The process of authorisation.

C.2.6.7.1 Pre-Conditions

- A template has been created in Data Distribution which specifies the information that the MDL could contain. The template defines the format for any category of data that could be required in the MDL.
- The authorisation process for MDL has been defined.
- Information Brokerage has been tasked with the information exchange requirement for the MDL.

C.2.6.7.2 View

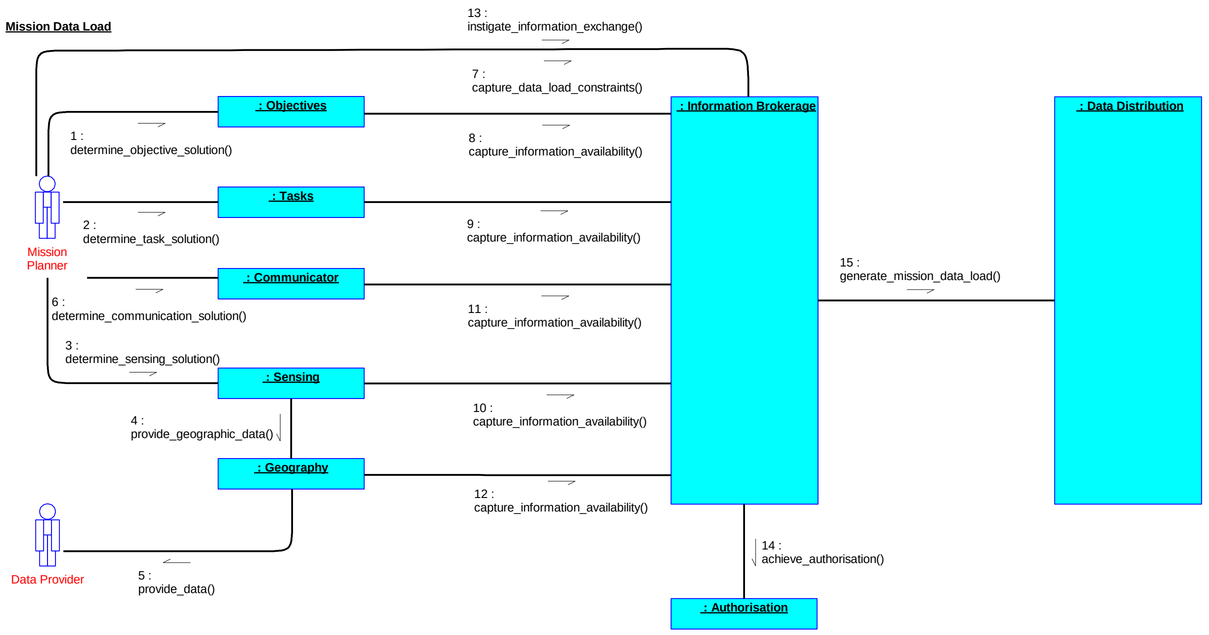

Mission Data Load
13 : instigate_information_exchange()
7 : capture_data_load_constraints()
: Objectives
: Information Brokerage
: Data Distribution
1 : determine_objective_solution()
8 : capture_information_availability()
: Tasks
2 : determine_task_solution()
9 : capture_information_availability()
Mission Planner
: Communicator
15 : generate_mission_data_load()
6 : determine_communication_solution()
11 : capture_information_availability()
3 : determine_sensing_solution()
: Sensing
4 : provide_geographic_data()
10 : capture_information_availability()
: Geography
12 : capture_information_availability()
Data Provider
5 : provide_data()
14 : achieve_authorisation()
: Authorisation

Figure 186: Mission Data Load IV

The Mission Planner interacts with the system to create a mission plan. The Mission Planner interacts with components in the Objective, Task, Action and Resource layers, as required (see the Control Architecture PYRAMID concept). This planning process results in the generation and refinement of mission planning data which is distributed across these components in accordance with their subject matter. This includes the data that will collectively form the MDL for the target platform defined by the Mission Planner. As the mission plan is populated Information Brokerage is informed of, and maintains a record of, available source data that is the subject of the information exchange requirement.

When required, the Mission Planner triggers the generation of the MDL and consequently Information Brokerage is instructed to enact the exchange solution. The Information Brokerage component instigates exchanges of the available information required to make up the MDL, taking into account the exchange requirements and the specific details of the mission plan. This includes any supporting data identified by components involved in the planning activity (a need for geographic data to support Sensing is shown as an example).

The Authorisation component undertakes the defined process steps to achieve authorisation. Once authorisation is confirmed Information Brokerage requests Data Distribution to compile the MDL in preparation for delivery.

C.2.6.7.3 Post-Conditions

- A complete MDL exists and is ready for formatting and transfer to the operational system.

C.2.6.7.4 Actors

Data Provider

The provider of data (e.g. geographical data) that is required to support the mission plan.

Mission Planner

An authorised operator of the mission planning system who is responsible for developing the mission plan.

C.2.6.8 Request for Information

An Operator makes a request for information (RFI). This request is for any EO or IR images of a specific physical location. This IV shows how the system is used to search for the requested images and then distribute them to meet the requirements of the RFI. This includes ensuring the format is correct and the clearance of the Image Receiver is appropriate for the classification of the images produced.

This example IV can be used to represent the retrieval of any type of mission generated data, by replacing the represented instance of the Sensor Products component with the appropriate component(s) responsible for the subject matter involved with the query, as each component is responsible for managing its own data including its associated storage.

It is assumed that:

- Images of the requested location have been produced.
- The Image Receiver is already known to the system, including aspects such as communication methods, clearance level, etc.
- The HMI Dialogue and Information Presentation have been configured for the Operator's role. This enables them to interact appropriately with the system.
- The system has been configured so that Information Brokerage is informed of the properties of images whenever the new image data are created (e.g. classification and time created).
- Authorisation to release images to the Image Receiver has been given as long as specific criteria are met.

The following considerations are excluded:

- How the images are distributed using the communications hardware.
- Interactions with storage.

C.2.6.8.1 Pre-Conditions

- A request for image data has been received.

C.2.6.8.2 View

Request for Information

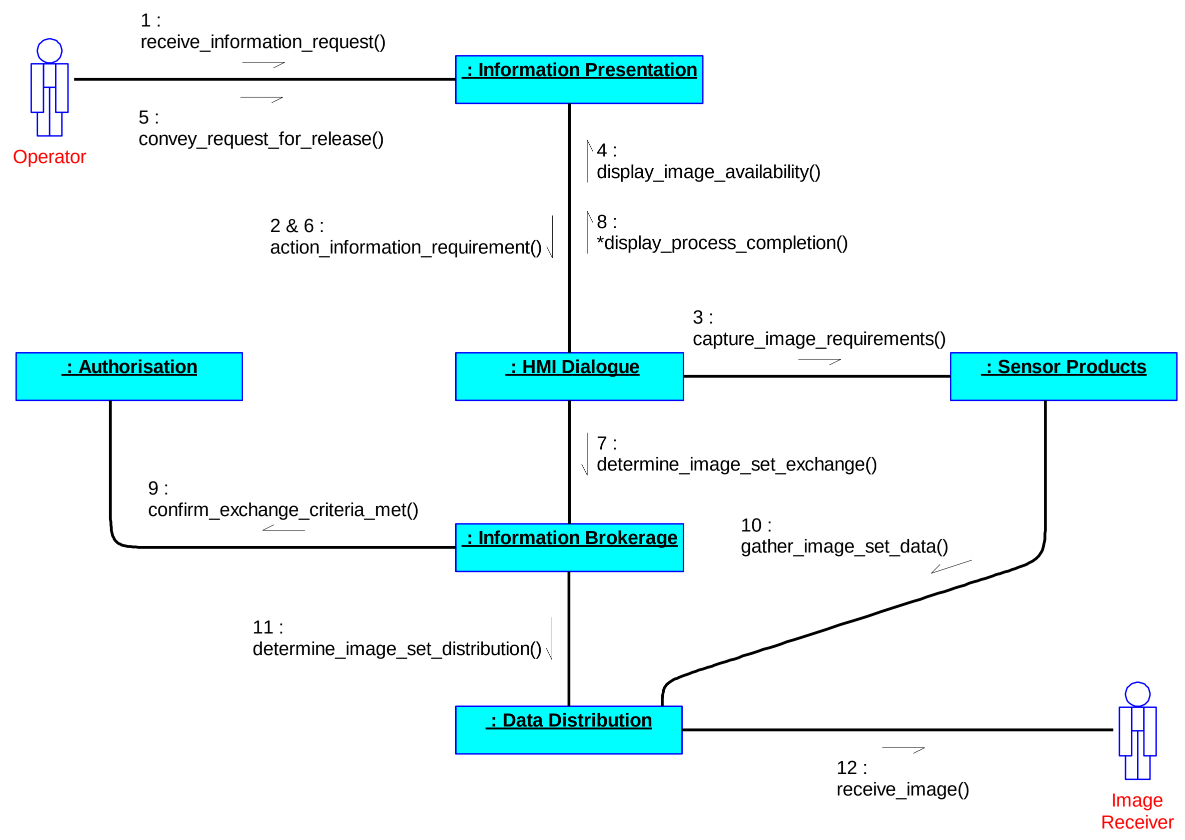

1 : receive_information_request()
5 : convey_request_for_release()
Operator
: Information Presentation
4 : display_image_availability()
2 & 6 : action_information_requirement()
8 : *display_process_completion()
3 : capture_image_requirements()
: Authorisation
: HMI Dialogue
: Sensor Products
7 : determine_image_set_exchange()
9 : confirm_exchange_criteria_met()
: Information Brokerage
10 : gather_image_set_data()
11 : determine_image_set_distribution()
: Data Distribution
12 : receive_image()
Image Receiver

Figure 187: Request for Information IV

To determine if the RFI can be met, the Operator will query the system to see if any stored images match the specified location. The Operator interacts to insert the search request via Information Presentation. HMI Dialogue then places a requirement on Sensor Products to provide an image of the specified location: this is only pulling on existing data. Sensor Products determines it holds images that meet the requested criteria. The Operator is then informed that the images exist, therefore the RFI can be met.

The Operator then decides that the images need to be released to the Image Receiver and inputs this request into the HMI via Information Presentation. HMI Dialogue responds to this request by requesting Information Brokerage to determine the exchange solution, to ensure the exchange is allowable and determine if any reclassification of data is required due to multiple data sources. Information Brokerage requests Authorisation to determine if the necessary criteria have been met and the exchange is allowable. Once authorisation is confirmed, Information Brokerage instigates the release of the image set by informing Data Distribution that the image set requires distribution to the Image Receiver. Data Distribution gathers the required data and determines a distribution solution, taking into account the communication resources as appropriate.

C.2.6.8.3 Post-Conditions

- The images have been released to the external system.

C.2.6.8.4 Actors

Operator

A user of the system, who has been given a role that allows them to search the system for data and instigate the release of the data.

Image Receiver

A system that is able to receive images.

C.2.6.9 Time Synchronisation between Nodes

This IV describes how time values may be synchronised between different nodes, each with access to a local time source. The nodes may be within an Exploiting Platform, or may be separate Exploiting Platforms.

It is assumed that:

- Transmission latency exists for communications between the nodes, though the latency is not large.

The following considerations are excluded:

- Alternative synchronisation strategies. The preferred strategy will depend on numerous factors, such as the relationship between nodes (e.g. connection type, distance, and interference), accuracy required, available time sources, and clock drift.
- Synchronisation between nodes where none are considered particularly reliable. In such cases a weighted average of principal times may be used to determine a reference time to be synchronised against.
- Mechanisms for communication of time information between nodes. See the Data Transfer IV for details.
- Identification of system health problems that may lead to a requirement for synchronisation.
- The setting of rules and limits concerning time sources or any allowable inconsistencies in time values.

C.2.6.9.1 Pre-Conditions

- A requirement for synchronisation between Node B and Node A has been identified.

C.2.6.9.2 View

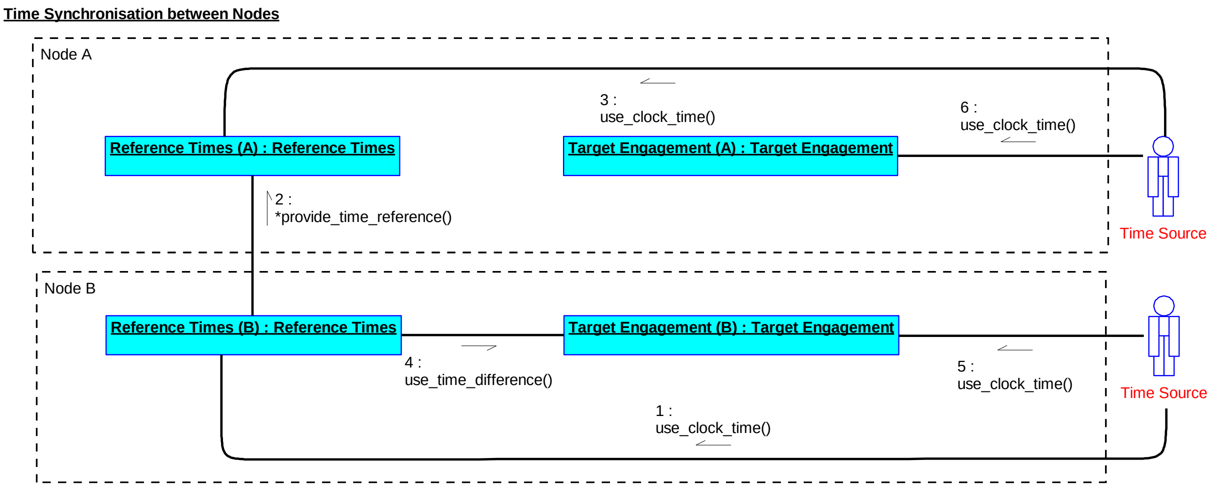

Time Synchronisation between Nodes
Node A
Reference Times (A) : Reference Times
Target Engagement (A) : Target Engagement
3 : use_clock_time()
6 : use_clock_time()
2 : *provide_time_reference()
Time Source
Node B
Reference Times (B) : Reference Times
Target Engagement (B) : Target Engagement
4 : use_time_difference()
5 : use_clock_time()
1 : use_clock_time()
Time Source

Figure 188: Time Synchronisation between Nodes IV

Reference Times (B) gets the local clock time from the local Time Source. Reference Times (B) then requests the current clock time Difference from Reference Times (A) - i.e. the difference between its provided local clock time instance and Node A's local clock time.

Reference Times (A) gets the local clock time from its local Time Source. Reference Times (A) determines time Difference between this value of clock time and that provided in the query by Reference Times (B) and returns the time Difference to Reference Times (B).

Reference Times (B) uses the provided time Difference, and any latencies in transactions and processing, to adjust (synchronise) its understanding of the equivalent Node A reference time.

Reference Times (B) provides the time Difference, now synchronised to Reference Times (A), to Target Engagement (B).

Target Engagement (B) consumes local clock times from its local Time Source, adjusted according to the time Difference information provided by Reference Times (B).

Target Engagement (A) consumes local clock times from its local Time Source.

C.2.6.9.3 Post-Conditions

- Use of time in Node B is synchronised to that of Node A.

C.2.6.9.4 Actors

Time Source

A time source, for example supplied by the local infrastructure.

C.2.6.10 Middleware Error

This group of interaction views illustrates how the PRA components can interact with middleware functionality in the context of health management. Middleware is software that provides a set of general purpose services for the PRA components, in a layered infrastructure architecture the middleware sits in between the application layer and the processing hardware layer.

The complexity of the middleware is dependent on the precise implementation, however it is expected that the middleware will be responsible for managing faults in the underlying processing hardware. The middleware will be able to take pre-defined actions in response to hardware faults, e.g. by restarting hardware or using backup resources.

The PRA health management components will interact with the middleware to perform fault handling functionality and will insulate other PRA components from such concerns. Other PRA components may have a limited role in managing the middleware, for example the Asset Transitions component triggering software configuration changes.

The PRA components will be responsible for understanding any change in their capability due to middleware changes.

C.2.6.10.1 Middleware Handled Error

This IV demonstrates how a hardware fault is identified and handled by the infrastructure. The PYRAMID components are not involved in the diagnosis and corrective activity; they are only informed of the loss of the resource, which may lead to a change in capability and a reduction in system redundancy. Information Presentation is used as an example of a component making use of a resource.

It is assumed that:

- The Middleware is able to handle certain types of fault arising from the underlying resource hardware.
- The Middleware understands the allowable software configurations, i.e. which components are permitted to use which resource.

The following considerations are excluded:

- Health Assessment components working and communicating with other Health Assessment components in a hierarchy, which would be relevant in more complex health scenarios.
- What Information Presentation is using the resource for.
- Any maintenance action to address the faults identified.
- Behaviour associated with running tests (e.g. CBIT).
- How system capability is determined based on a published anomaly or health assessment.

C.2.6.10.1.1 Pre-Conditions

- No pre-conditions identified.

C.2.6.10.1.2 View

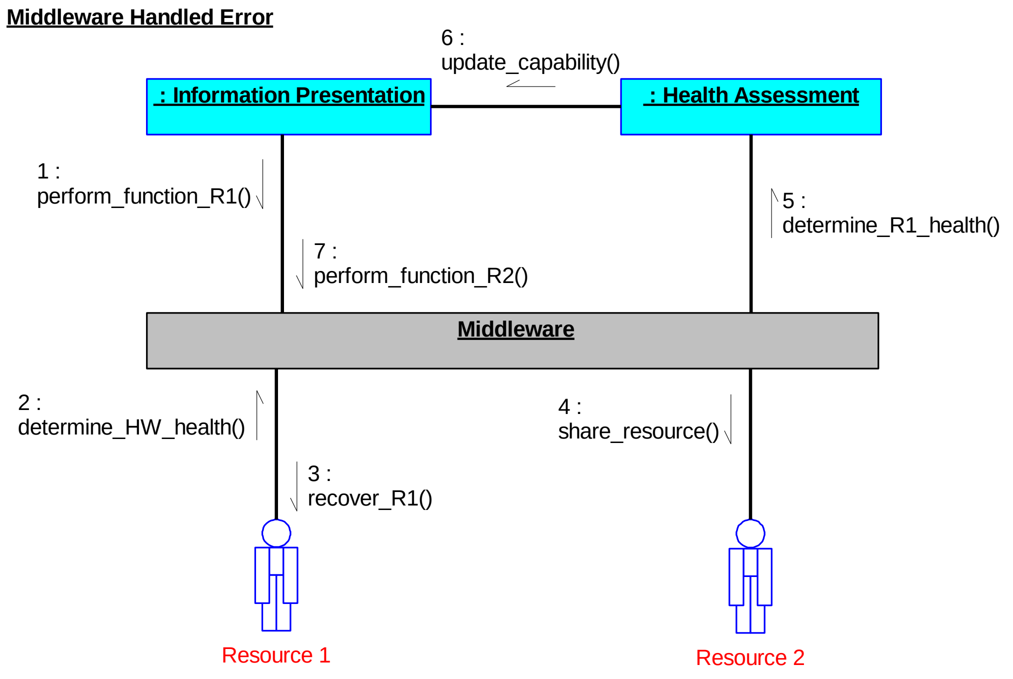


Figure 189: Middleware Handled Error IV

Information Presentation is using Resource 1 and performing its function when Resource 1 identifies a fault during CBIT and reports it to the Middleware. The fault falls within the remit of what the Middleware is permitted to address and the Middleware attempts to recover Resource 1, e.g. by performing a warm start. The recovery action is not successful (i.e. the fault is still present) therefore the Middleware updates the configuration to allow Information Presentation to use Resource 2 (albeit on a shared basis) instead of Resource 1.

The Middleware reports the appropriate details of the fault and resource remapping to Health Assessment, which logs the information to assist subsequent maintenance. Health Assessment then informs Information Presentation that there may be a change to its capability due to the increase in response times, this is due to Resource 2 being a shared resource; the fact that the resource has changed is invisible to Information Presentation, which continues to use the available resource in the conduct of its activities.

C.2.6.10.1.3 Post-Conditions

- The Middleware has remapped the resource allocation from Resource 1 to Resource 2.
- Health Assessment has logged the fault identified by Resource 1 for subsequent maintenance action.

C.2.6.10.1.4 Actors

Resource 1

A hardware processing resource used by the system.

Resource 2

A hardware processing resource used by the system.

C.2.6.10.2 PRA Handled Error

This IV demonstrates how a hardware fault is identified and handled by the PYRAMID components which then request the Middleware to reconfigure to allow access to an alternative resource. The PYRAMID components also deal with the change in capability and reduction in system redundancy. Information Presentation is used as an example of a component making use of a resource.

It is assumed that:

- The Middleware understands the allowable software configurations, i.e. which components are permitted to use which resource.
- The Middleware is unable to handle the resource failure.
- BIT can be run on the hardware resource with no adverse impact on ongoing activities.

The following considerations are excluded:

- What Information Presentation is using the resource for.
- Any maintenance action to address the faults identified.
- How system capability is determined based on a published anomaly or health assessment.
- Cyber attack assessment using Cyber Defence.

C.2.6.10.2.1 Pre-Conditions

- No pre-conditions identified.

C.2.6.10.2.2 View

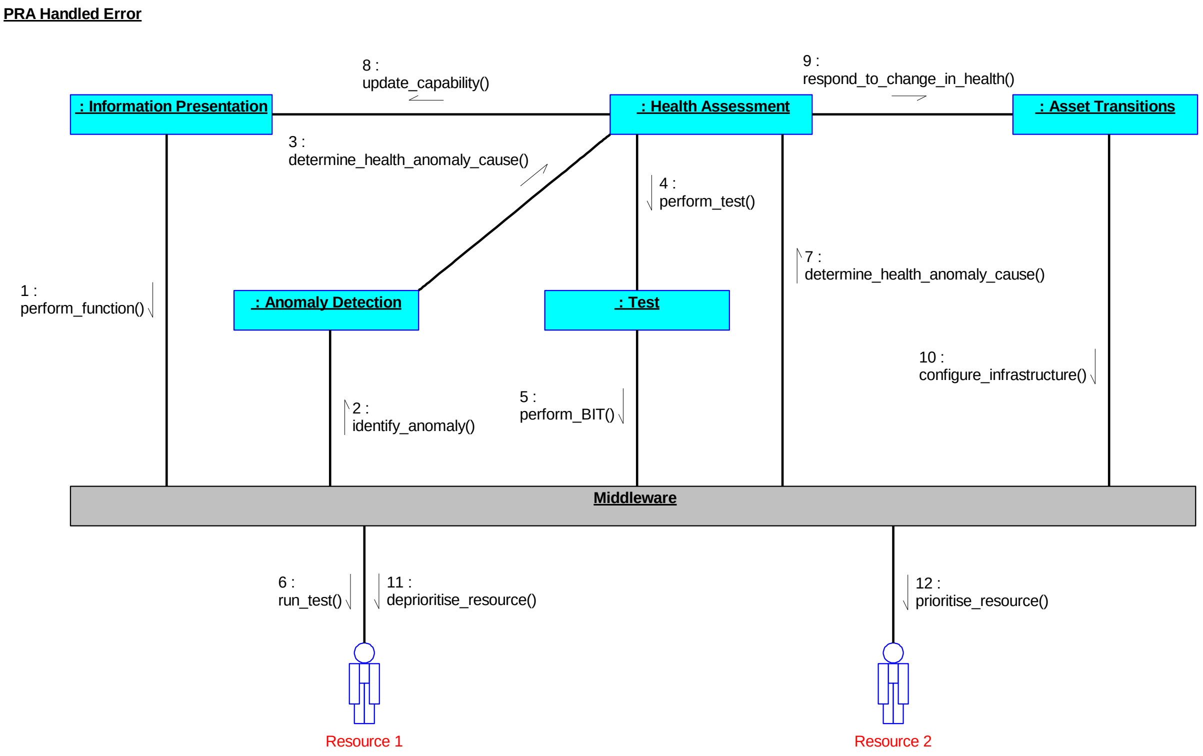


Figure 190: PRA Handled Error IV

Information Presentation is running on Resource 1 and performing its function. Anomaly Detection identifies a difference between the expected and actual state of the system and flags this as an anomaly to Health Assessment (also sent to Cyber Defence for investigation of whether this is indicative of a cyber attack - not shown here). Health Assessment identifies that there is insufficient information available to determine the cause of the anomaly and therefore directs Test to gather more information. Test instructs the middleware to run a BIT on Resource 1 and the test results indicate that there is a fault. The test results are passed to Health Assessment which in turn communicates this health change to Information Presentation, Information Presentation determines that it now has a reduced capability. The change in health identified by Health Assessment is also provided to Asset Transitions, which instructs the Middleware to use Resource 2 instead of Resource 1. Information Presentation no longer has a reduced capability.

C.2.6.10.2.3 Post-Conditions

- Information Presentation is running on Resource 2.
- Health Assessment has logged the fault identified in Resource 1 for subsequent maintenance action.

C.2.6.10.2.4 Actors

Resource 1

A hardware processing resource used by the system.

Resource 2

A hardware processing resource used by the system.

C.2.6.10.3 PRA Handled Error - Distributed Hardware

This IV demonstrates how a hardware fault in a primary resource is identified by the infrastructure, resulting in a handover to a secondary resource commanded by the PYRAMID components, which also handle any change in capability and reduction in system redundancy. Information Presentation is used as an example of a component making use of a resource.

It is assumed that:

- The Middleware is able to handle certain types of fault arising from the underlying resource hardware.
- The Middleware understands the allowable software configurations, i.e. which components are permitted to use which resource.

The following considerations are excluded:

- What Information Presentation is using the resource for.
- Any maintenance action to address the faults identified.
- Behaviour associated with running tests (e.g. CBIT).
- How system capability is determined based on a published anomaly or health assessment.
- Cyber attack assessment using Cyber Defence.

C.2.6.10.3.1 Pre-Conditions

- No pre-conditions identified.

C.2.6.10.3.2 View

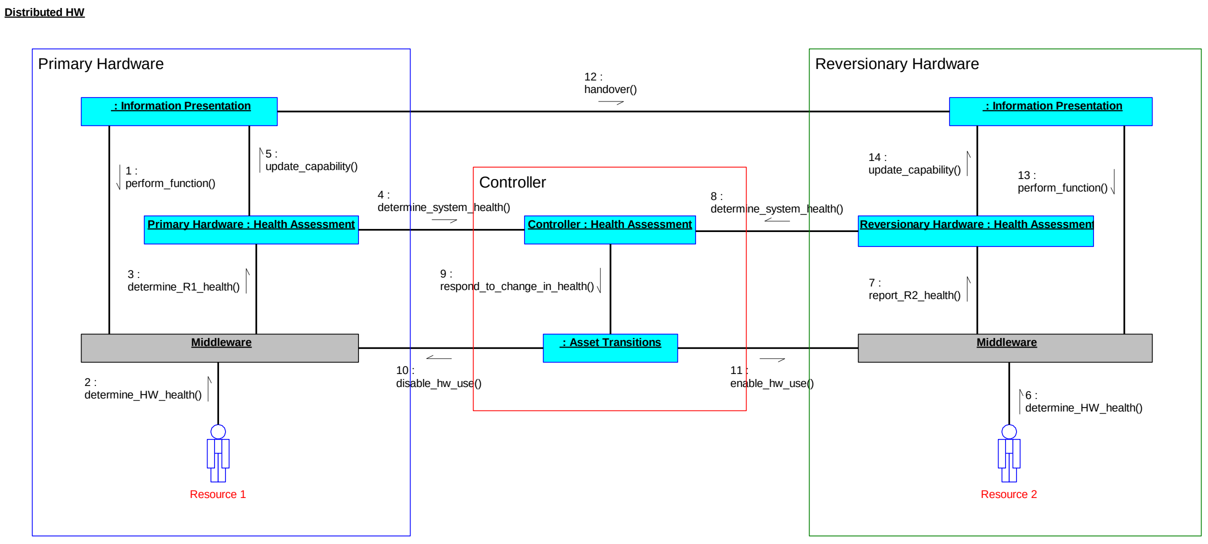

Distributed HW
Primary Hardware
Reversionary Hardware
Controller
: Information Presentation
: Information Presentation
Primary Hardware : Health Assessment
Controller : Health Assessment
Reversionary Hardware : Health Assessment
: Asset Transitions
Middleware
Middleware
Resource 1
Resource 2
1 : perform_function()
2 : determine_HW_health()
3 : determine_R1_health()
4 : determine_system_health()
5 : update_capability()
6 : determine_HW_health()
7 : report_R2_health()
8 : determine_system_health()
9 : respond_to_change_in_health()
10 : disable_hw_use()
11 : enable_hw_use()
12 : handover()
13 : perform_function()
14 : update_capability()

Figure 191: Distributed HW IV

Information Presentation is running on Resource 1 and performing its function when Resource 1 identifies a fault during CBIT and reports it to the Middleware. The Middleware reports the appropriate details of the fault to the Primary Hardware instance of Health Assessment, which logs the information to assist subsequent maintenance. The change in health is available to the Controller instance of Health Assessment.

The Controller instance of Health Assessment also has the health status reported by the Reversionary Hardware instance of Health Assessment. The Controller instance of Health Assessment reports the combined health status of Resource 1 and Resource 2 to Asset Transitions. Asset Transitions then coordinates the switch over from the Primary Hardware to the Reversionary Hardware by communicating with the related Middleware, which disables the use of Resource 1 and enables the use of Resource 2. The two instances of Information Presentation share information which allows them to perform a 'hot swap' handover when the switch in hardware is triggered. The Information Presentation running on the Reversionary Hardware is now able to function with the same capability prior to the fault on Resource 1.

C.2.6.10.3.3 Post-Conditions

- Information Presentation is running on Reversionary Hardware (Resource 2).
- Health Assessment has logged the fault identified by Resource 1 for subsequent maintenance action.

C.2.6.10.3.4 Actors

Resource 1

A hardware processing resource used by the system.

Resource 2

A hardware processing resource used by the system.

C.2.7 Operator Interactions Views

This section contains IVs relating to the interaction of a system with human operators.

The Figure 192: Operator Interaction Views UC Diagram shows all the use cases within the Operator Interaction Views and any actors that interact with them.

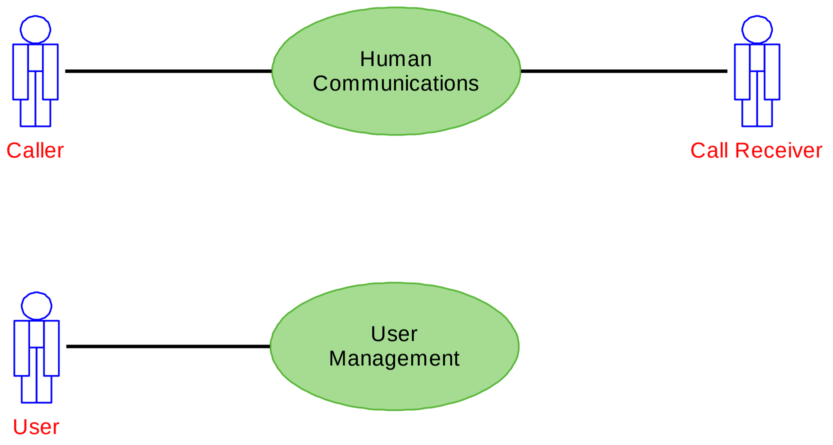


Figure 192: Operator Interaction Views UC Diagram

C.2.7.1 Human Communications

This covers the initiation and making of a phone call between two humans via the system. The behaviour is the same if it is between two authorised operators or an authorised operator and an external entity.

Note: While usually a phone call consists of four unidirectional streams (audio out, audio back, call stats out, and call stats back), only one is shown in Figure 193: Human Communications IV; the understanding of the relationship between the streams is in Information Brokerage, not in Data Distribution (which just sees four channels). These interactions are consistent with normal communication exchange protocols.

Similar methods can support other real-time communication, such as video streams. Where real-time communication is not needed, transfers can be handled in the same way as other machine-to-machine communication and do not require any special behaviour.

The closing down of a communication session can be processed as the reverse of opening a session.

Note that while both the initiating and receiving systems are PYRAMID based in this example, this does not need to be the case.

It is assumed that:

- The correct devices (e.g. headsets) are available to both call parties and the state of these are known.
- All call parties (initiating and receiving users) are logged on and are available.
- The location and registration details have been captured by Human Interaction, as provided by other components.
- All call parties are authorised to communicate with each other.

The following considerations are excluded:

- The setup, and use of, the communications infrastructure to achieve the call.
- Data transfer for inter-node interactions, between component instances at the initiating end and receiving end.
- Conference calls, as the build set is the same, except with multiple Callers and Call Receivers. Data Distribution also provides a "consolidated output", instead of a simple "output".
- Some steps have been removed to reduce cluttering, such as confirmation events (like the Receiving Human Interaction informing the Initiating Human Interaction that the call has been accepted).

C.2.7.1.1 Pre-Conditions

- Connectivity between the users is in place.

C.2.7.1.2 View

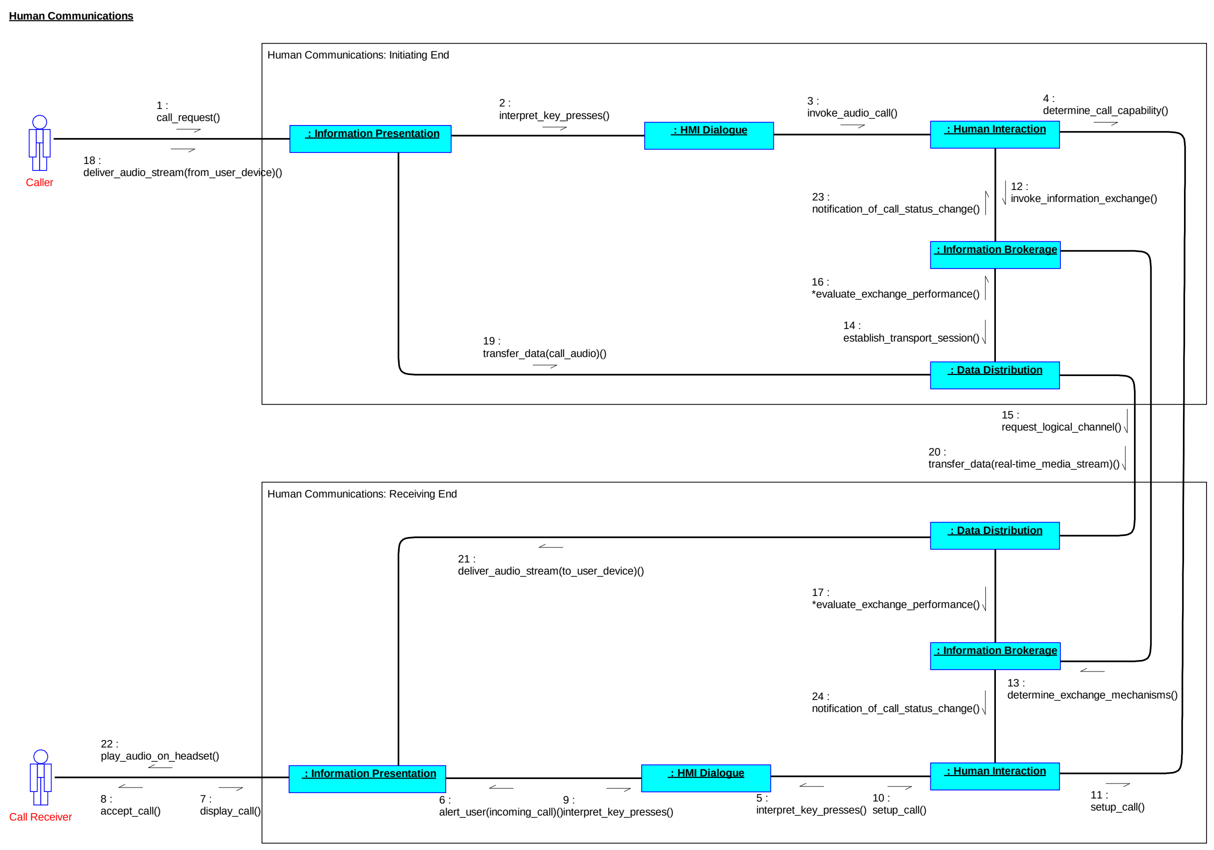

Human Communications
Human Communications: Initiating End
Caller
1 : call_request()
18 : deliver_audio_stream(from_user_device)()
: Information Presentation
2 : interpret_key_presses()
: HMI Dialogue
3 : invoke_audio_call()
: Human Interaction
4 : determine_call_capability()
23 : notification_of_call_status_change()
12 : invoke_information_exchange()
: Information Brokerage
16 : *evaluate_exchange_performance()
14 : establish_transport_session()
19 : transfer_data(call_audio)()
: Data Distribution
15 : request_logical_channel()
20 : transfer_data(real-time_media_stream)()
Human Communications: Receiving End
: Data Distribution
21 : deliver_audio_stream(to_user_device)()
17 : *evaluate_exchange_performance()
: Information Brokerage
13 : determine_exchange_mechanisms()
24 : notification_of_call_status_change()
22 : play_audio_on_headset()
Call Receiver
: Information Presentation
: HMI Dialogue
: Human Interaction
8 : accept_call()
7 : display_call()
6 : alert_user(incoming_call)()
9 : interpret_key_presses()
5 : interpret_key_presses()
10 : setup_call()
11 : setup_call()

Figure 193: Human Communications IV

A Caller requests an audio call with a Call Receiver via Information Presentation. Information Presentation interprets the key presses to be a number sequence and passes the information onto HMI Dialogue. HMI Dialogue interprets the sequence from the Caller as a request to dial a Call Receiver at the other end.

HMI Dialogue connects the request to Human Interaction via a request for an audio call.

Human Interaction determines the location and availability of the Call Receiver. Human Interaction (initiating end) then requests an audio call with Human Interaction (receiving end) after checking that the required devices exist at both call endpoints. This is a negotiation to start a call, between the two identified users and includes any agreement that the equipment required for a communication of that type is available. Human Interaction (receiving end) requests a call action from HMI Dialogue to connect call devices to the communication delivery system.

A call alert is sent to the Call Receiver, making the phone ring to alert them that someone wants to interact with them. When the call is accepted via a key press received by Information Presentation, and understood by HMI Dialogue, HMI Dialogue requests the set-up of the audio call.

Human Interaction (initiating end) requests that Information Brokerage sets-up the call (a real-time interaction based service level agreement). The two instances of Information Brokerage determine the available exchange mechanisms between the two locations. This includes how the audio will be digitally represented (encoded) and what exchange method will be used. Having decided on how the exchange is to be established, Information Brokerage requests Data Distribution to establish an exchange protocol and start a session for the audio. Data Distribution requests a logical channel to be opened from the relevant comms components (not shown) between the two locations, allowing the call to take place end-to-end.

Audio from the user is digitally encoded into a data stream by the user's device and Information Presentation, and connected to Data Distribution for delivery between each end.

Note: This is shown unidirectionally, but in reality will be bi-directional.

During the call Data Distribution reports traffic statistics to Information Brokerage which monitors the quality of the service against the requested characteristics; if the quality falls below the acceptable levels Human Interaction is notified; other than that Human Interaction is not involved again until call teardown.

C.2.7.1.3 Post-Conditions

- Call Receiver and Caller are in a call with each other and able to communicate directly.

C.2.7.1.4 Actors

Call Receiver

A user that has access to appropriate devices for the purpose of communicating with another user, and another user wishes to contact them.

Caller

An authorised user that has access to appropriate devices for the purpose of communicating with another user, and wishes to start communication.

C.2.7.2 User Management

The User needs to log into their operator workstation in order to perform their operational role. The operator workstation is configured with two parts in this example. Information Presentation is used to enter a username and password. When the User logs in, Information Presentation (Login) becomes hidden and Information Presentation (Flight Reference) is enabled. This will show display elements (e.g. altitude) in line with the assigned User role. HMI Dialogue is configured by the assignment of the User's role to provide the appropriate data.

In this scenario, user login is handled through a username and password combination. Similar methods can be used to support other mechanisms of login (e.g. voice recognition, or using a hardware token).

It is assumed that:

- The User must first log into the operating system, then log into the PYRAMID system separately. A deployment may use the operating system to provide User Accounts functionality.
- This scenario is for a self-contained system, therefore no credentials are transmitted, and so encryption prior to interaction with User Accounts is not needed.

The following considerations are excluded:

- User account administration such as creation and deletion of accounts, assignment of administrator privileges or recovery of passwords.
- Logging into the operating system; this scenario is only concerned with logging into the PYRAMID system.
- Dynamic role allocation.
- Presentation of information.
- User input processing.

C.2.7.2.1 Pre-Conditions

- The User is not logged in to the PYRAMID system.
- The User has a user account configured in User Accounts.
- The User has initial roles configured in User Roles.

C.2.7.2.2 View

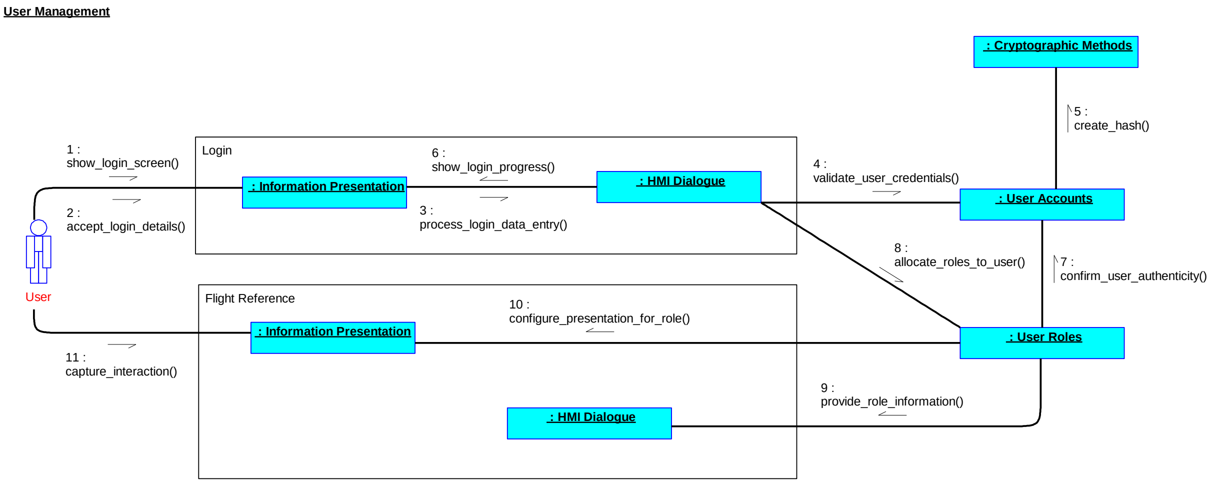

User Management
: Cryptographic Methods
5 : create_hash()
Login
1 : show_login_screen()
6 : show_login_progress()
4 : validate_user_credentials()
: Information Presentation
: HMI Dialogue
: User Accounts
2 : accept_login_details()
3 : process_login_data_entry()
8 : allocate_roles_to_user()
7 : confirm_user_authenticity()
User
Flight Reference
10 : configure_presentation_for_role()
: Information Presentation
: User Roles
11 : capture_interaction()
9 : provide_role_information()
: HMI Dialogue

Figure 194: User Management IV

The User requests display of the login screen and Information Presentation (Login) makes it visible.

The User enters their username and password to request login, which is accepted by Information Presentation (Login).

HMI Dialogue (Login) creates a login request using the username, operator workstation identifier and password, then passes it to User Accounts.

User Accounts validates the credentials in the login request and informs the HMI Dialogue (Login) that the User has successfully logged in. User Accounts requests that Cryptographic Methods generate a hash of the entered password information.

HMI Dialogue (Login) requests User Roles to assign the User's the initial roles.

User Roles requests User Accounts to confirm the User's authenticity. Once confirmed, User Roles determines the User's initial roles.

User Roles informs the HMI Dialogue (Flight Reference) and Information Presentation (Flight Reference) of the roles associated with the User at the operator workstation.

Information Presentation (Flight Reference) updates the presentation to tailor it to the allocated roles (e.g. enabling, disabling, showing or hiding presentation elements).

HMI Dialogue (Flight Reference) is configured based on the User's role to provide the appropriate data.

The User interacts with the system to select the information they want to see and Information Presentation (Flight Reference) determines the appropriate elements to display.

Note: Within this IV HMI Dialogue and Information Presentation have been shown split apart to enable clarity of the interactions between those required for the Login process and those for the tailoring of the display and required data for a specific role (e.g. Flight Reference).

C.2.7.2.3 Post-Conditions

- The User is logged into the operator workstation and has been allocated initial roles (they are now an authorised operator).
- The Information Presentation and HMI Dialogue components have been configured for the User's allocated roles.

C.2.7.2.4 Actors

User

A person with an existing user account who needs to log into the system.

C.2.8 Decision Making Views

This section contains IVs relating to high-level decision making and authorisation within a system.

The Figure 195: Decision Making Views UC Diagram shows all the use cases within the Decision Making Views and any actors that interact with them.

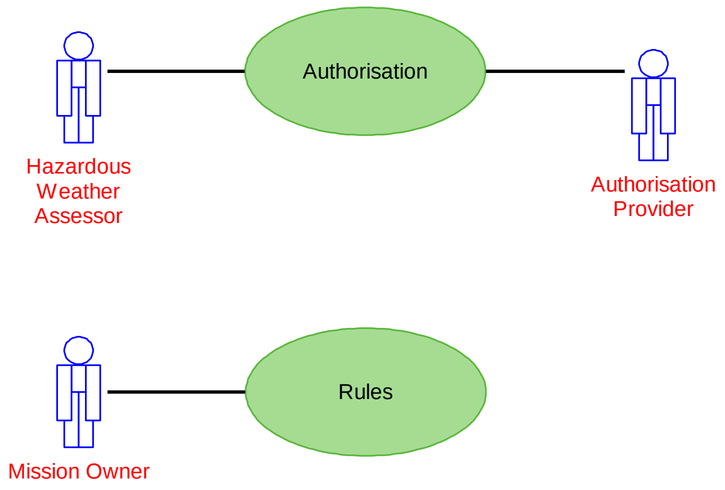


Figure 195: Decision Making Views UC Diagram

C.2.8.1 Authorisation

The vehicle is following a planned route which has adverse weather along it. Whilst executing the route, the vehicle encounters the adverse weather. This means the current routing solution meets avoidance criterion and a new route, which crosses restricted airspace, is required. This IV illustrates how authorisation is requested and granted for a route that crosses restricted airspace.

It is assumed that:

- The deployment has a level of autonomy that allows Routes to determine a routing solution.
- Routes that cross restricted airspace require authorisation by an authorised operator.
- Routes considers all relevant aspects of the SA picture when determining that a new route crossing restricted airspace is necessary to avoid the hazardous weather.

The following considerations are excluded:

- How routes are enacted.
- The way that weather data is detected and processed.
- How the authorised operator's situation awareness is facilitated via the HMI.
- How authorisation requests are presented to the authorised operator via the HMI.
- How authorisation is granted by the authorised operator via the HMI.

C.2.8.1.1 Pre-Conditions

- A planned route is being executed.
- Operational Rules and Limits has been loaded with the rules for routing in different airspaces.

- Authorisation has been loaded with an authorisation policy, which contains rules for the authoriser roles allowed to give authorisations for crossing a restricted airspace.

C.2.8.1.2 View

Authorisation

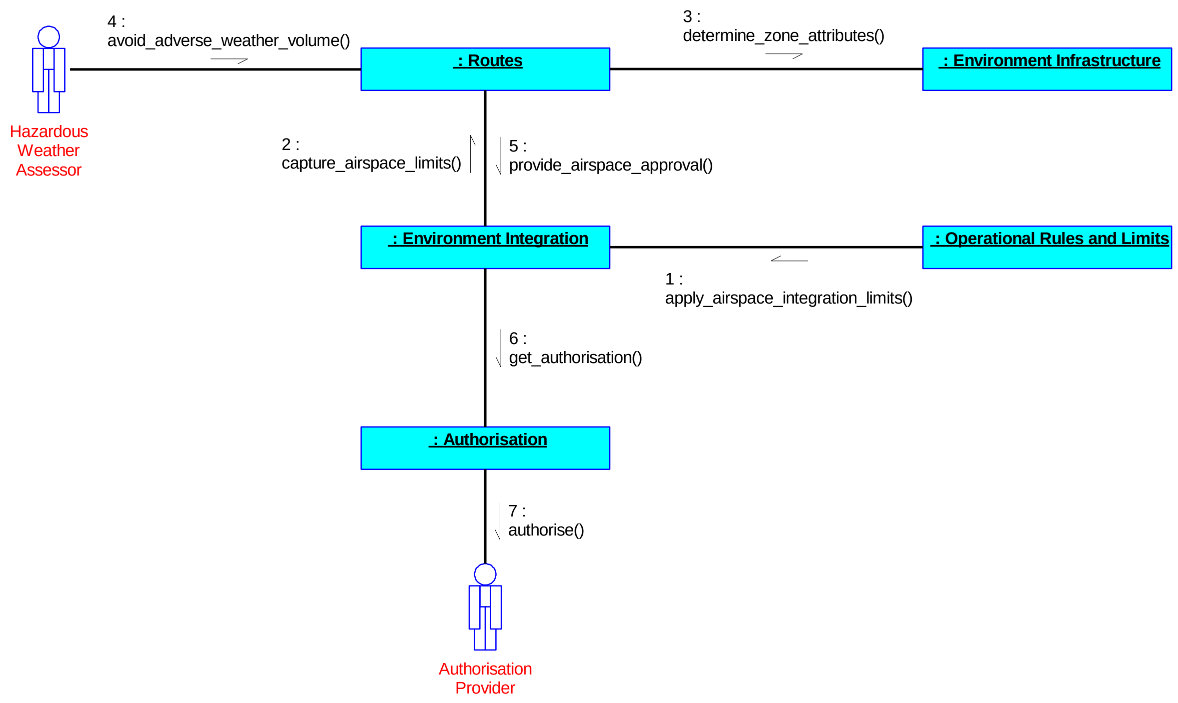

4 : avoid_adverse_weather_volume()
: Routes
3 : determine_zone_attributes()
: Environment Infrastructure
Hazardous Weather Assessor
2 : capture_airspace_limits()
5 : provide_airspace_approval()
: Environment Integration
: Operational Rules and Limits
1 : apply_airspace_integration_limits()
6 : get_authorisation()
: Authorisation
7 : authorise()
Authorisation Provider

Figure 196: Authorisation IV

Operational Rules and Limits applies limits on Environment Integration. These limits act as a constraint that prohibits flight though restricted airspace unless authorisation is granted; it remains in effect until authorisation is obtained, at which point it becomes inapplicable.

Environment Integration requests Routes to incorporate the airspace limits as part of the routing solution, in accordance with the constraint set by Operational Rules and Limits. Using identifiers which relate to airspace volumes that are provided by Environment Integration, Routes consults Environment Infrastructure to retrieve the properties relating to these airspace volumes. Environment Infrastructure provides the necessary information.

The Hazardous Weather Assessor identifies an adverse weather event and demands that Routes avoids the affected volume of airspace. Routes has to change the current routing solution accordingly and requests that Environment Integration obtain the authorisation required to accommodate the change, considering the constraint set by Operational Rules and Limits.

Environment Integration requests Authorisation to gain the authorisation needed to fly through a restricted volume of airspace in order for the vehicle to avoid the adverse weather. Authorisation determines that to fly though the identified airspace, authorisation is required from the Authorisation Provider. The Authorisation Provider authorises the request, and on this occasion permits the temporary use of the restricted airspace. Authorisation provides authorisation to Environment Integration allowing for the constraint to be temporarily not applicable. Routes is notified that authorisation has been granted for the vehicle to fly through the restricted volume of airspace, and it is confirmed that the revised route can be achieved.

C.2.8.1.3 Post-Conditions

- The new route has been authorised.

C.2.8.1.4 Actors

Authorisation Provider

An authorised operator of the system who is able to authorise a route change through restricted airspace.

Hazardous Weather Assessor

The Hazardous Weather Assessor is responsible for detecting weather events and determining their threat to the survival of the aircraft.

C.2.8.2 Rules

This IV illustrates how a PYRAMID deployment could interpret rules in order to determine the zones within which it is not allowed to fly, and which therefore must be identified and the appropriate environmental rules determined before any routing action can be performed.

There are a number of different categories of rules which an Exploiting Platform might need to comply with. Included within these are Rules of Engagement, EMCON policies and behavioural rules, however other rule sources may exist. This IV will focus on rules which limit where the system can fly.

It is assumed that:

- The system is operating in a benign environment with guaranteed communications between the system and the authorised operator.

The following considerations are excluded:

- Operating the system in accordance with the calculated limits.
- Restrictions on where the system can operate which are not derived from rules.
- Replacement of rules and subsequent re-calculation of limits.

C.2.8.2.1 Pre-Conditions

- The relationship between rules and system limits has been defined.
- A set of rules with conditions defining when they apply has been defined.

C.2.8.2.2 View

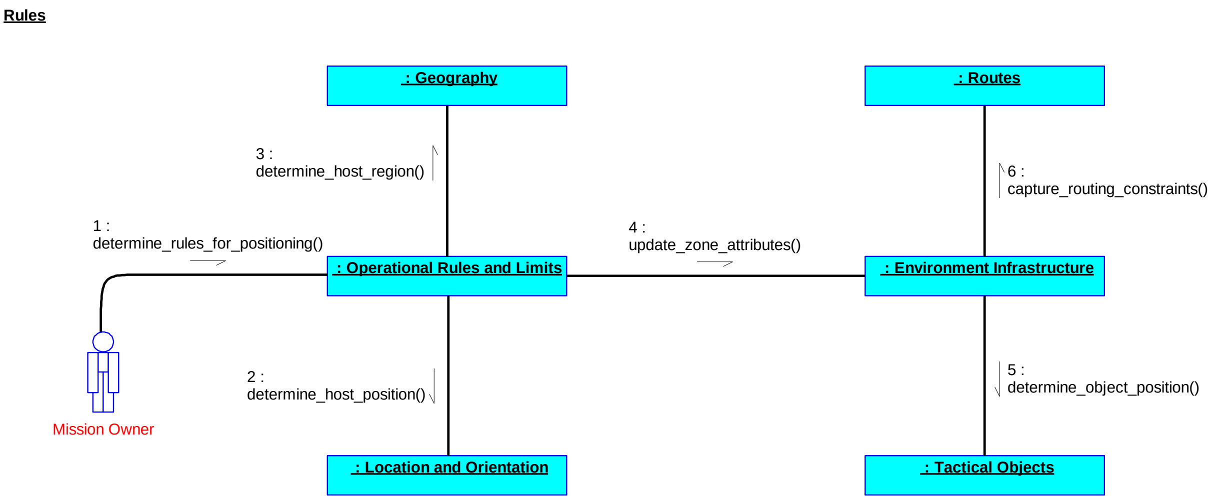

Rules
: Geography
: Routes
3 : determine_host_region()
6 : capture_routing_constraints()
1 : determine_rules_for_positioning()
4 : update_zone_attributes()
: Operational Rules and Limits
: Environment Infrastructure
2 : determine_host_position()
5 : determine_object_position()
Mission Owner
: Location and Orientation
: Tactical Objects

Figure 197: Rules IV

The Mission Owner sets the context for the mission, Operational Rules and Limits obtains ownship position from Location and Orientation and then requires Geography to identify the geographical region ownship is currently within. With the context and the region Operational Rules and Limits determines the currently applicable rules and thus the current limits on the system.

These limits are communicated to Environment Infrastructure which uses them to update the attributes of environmental regions it understands (e.g. active regions of restricted airspace). The position of regions within Geography are not affected by the limits.

For limits associated with tactical objects (e.g. the missile engagement zone around a SAM system), Environment Infrastructure will obtain the object positions from Tactical Objects and using the limits from Operational Rules and Limits will determine appropriate regions to implement these limits.

Environment Infrastructure will provide all regions as constraints to Routes which will account for them when determining the route.

C.2.8.2.3 Post-Conditions

- The limits on where the system is permitted to operate are understood.

C.2.8.2.4 Actors

Mission Owner

The 'owner' of the mission who sets the context within which the system is operating.

C.2.9 Offensive Actions Views

This section contains IVs relating to the offensive capability of a system and how it is used.

The Figure 198: Offensive Actions Views UC Diagram shows all the use cases within the Offensive Actions Views and any actors that interact with them.

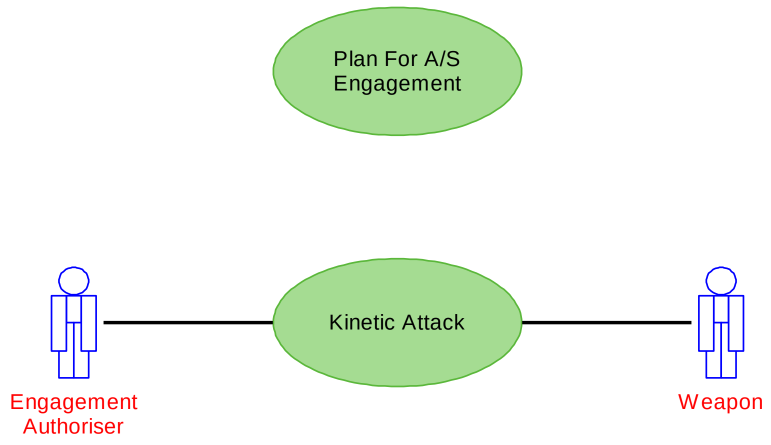


Figure 198: Offensive Actions Views UC Diagram

C.2.9.1 Plan For A/S Engagement

This IV illustrates the planning of a single engagement, which may include a combination of different weapons. The target has already been identified and selected for attack. This IV shows determination of the weapon package and the route to the release location, and activities to improve the target position accuracy. This release location is dependent on a number of factors including the weapon type and weather.

Planning for the deployment of a non-weapon store, such as a deployable sensor or cargo store, would be extremely similar and use largely the same components (aside from a different Tactics extension and not using Destructive Effects or Susceptibility).

It is assumed that:

- The weapons available for the A/S engagement are unpowered and have no routing capability.

The following considerations are excluded:

- All activities relating to the actual execution of the A/S engagement, as shown in the Kinetic Attack IV, such as weapon power up, dynamic LAR refinement during engagement execution, or execution of a route to a launch position.
- Setting weapon parameters, including weapon sensor parameters.
- Determination of the target in accordance with extant rules (e.g. RoE or targeting directive).
- Full determination of weapon availability and serviceability.
- Full reservation process of weapons as resources.
- Full interactions required to determine the weapon settings necessary to achieve the required weapon effect.

- Communication components, shown in the Data Transfer IV, since different weapons require different messaging systems.
- Power requirements, shown in the Power Management IV.
- Authorisation to commence planning the attack.

C.2.9.1.1 Pre-Conditions

- Weapon systems are fitted and able to communicate with the platform as applicable.
- The target has been identified and selected for attack.
- All necessary engagement tasking parameters (e.g. constraints, budgets, and criteria) are known.

C.2.9.1.2 View

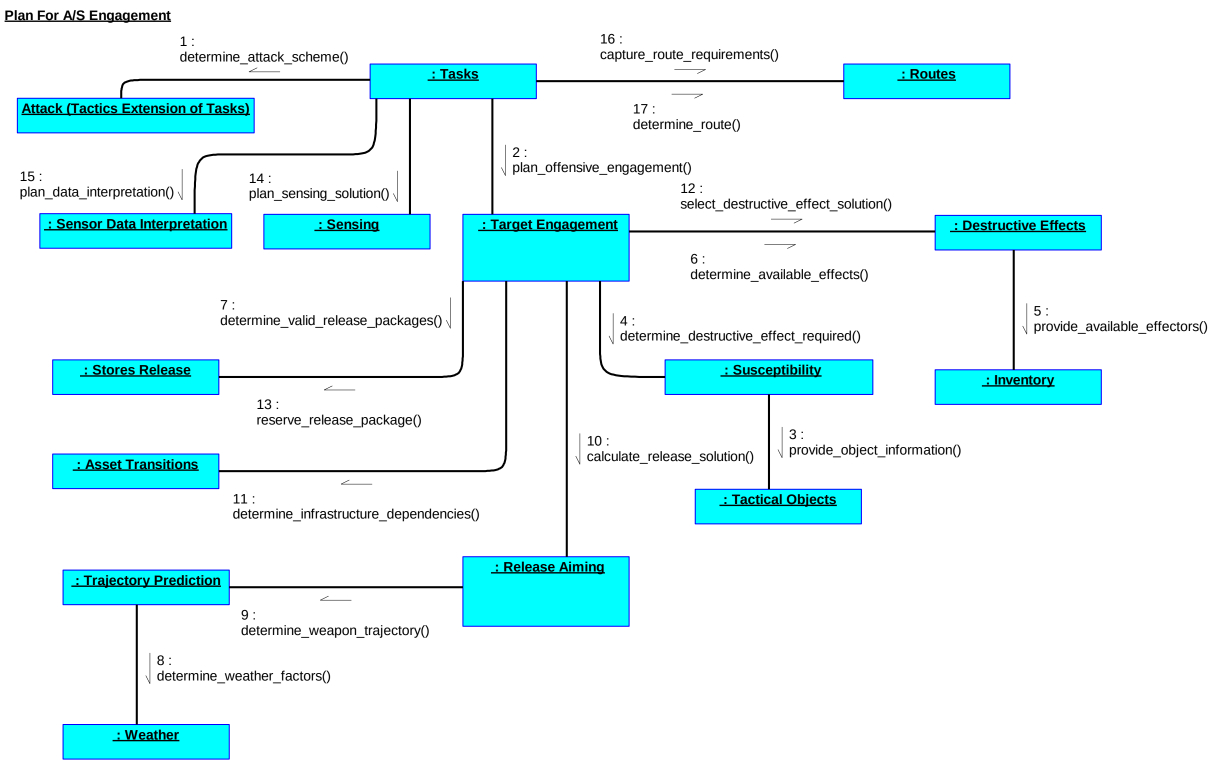

Plan For A/S Engagement
: Tasks
: Routes
Attack (Tactics Extension of Tasks)
: Sensor Data Interpretation
: Sensing
: Target Engagement
: Destructive Effects
: Stores Release
: Susceptibility
: Inventory
: Asset Transitions
: Tactical Objects
: Trajectory Prediction
: Release Aiming
: Weather
1 : determine_attack_scheme()
2 : plan_offensive_engagement()
3 : provide_object_information()
4 : determine_destructive_effect_required()
5 : provide_available_effectors()
6 : determine_available_effects()
7 : determine_valid_release_packages()
8 : determine_weather_factors()
9 : determine_weapon_trajectory()
10 : calculate_release_solution()
11 : determine_infrastructure_dependencies()
12 : select_destructive_effect_solution()
13 : reserve_release_package()
14 : plan_sensing_solution()
15 : plan_data_interpretation()
16 : capture_route_requirements()
17 : determine_route()

Figure 199: Plan For A/S Engagement IV

The Attack extension to Tasks is used to determine the appropriate scheme for prosecuting targets by coordinating dissimilar but mutually supporting actions.

Tasks requests that Target Engagement plan the engagement (this is one of the actions that needs to be planned to prosecute the attack task). Target Engagement then begins to plan the offensive engagement.

Susceptibility gathers information about the target using information provided by Tactical Objects and determines the required effects needed to exploit the target's vulnerabilities. Target Engagement requests Destructive Effects to determine what destructive effects the air vehicle has at its disposal to satisfy these effect requirements. Using the available weapon information provided by Inventory, Destructive Effects determines options that could achieve the required destructive effect (in this case where each effect package option is comprised of a set of weapon effectors).

Target Engagement then determines the feasibility of each effect package option and selects the most suitable one. To do this for each effect package option it establishes:

- If the weapons associated with a package will be releasable (determined by Stores Release which constructs a potential release package).
- If the weapons associated with a package are sufficiently serviceable to prosecute a mission (the interactions associated with this are not shown).
- If there is an acceptable release region (determined by Release Aiming).

Since the release aiming solution is influenced by the weapon settings required to achieve the desired effect (such as impact angle), Destructive Effects provides the settings applicable to each effects package option. (The interactions involving information that Destructive Effects needs to establish this are not shown.)

Trajectory Prediction determines the predicted trajectory of the weapon release, taking into account atmospheric conditions provided by Weather, and provides this to Release Aiming. Using the predicted trajectory information, Release Aiming calculates the aiming solution.

Asset Transitions determines a transition solution to meet dependencies related to activities planned by Target Engagement (such as determining weapon activation activities).

Once Target Engagement has determined the best solution the selected plan is disseminated to the relevant components satisfying the different parts of the solution:

- The chosen destructive effects option is indicated to Destructive Effects so that it can apply the associated settings when the plan is enacted.
- Stores Release is requested to select the required release package and Stores Release then arranges for the relevant weapons to be reserved (the reservation interactions are not shown). Destructive Effects is informed of the chosen weapons package so that it can apply the settings to the appropriate weapons.

Target Engagement informs Tasks whether it has a solution for the engagement that meets the requirements and indicates the solution costs and any solution dependencies needed to achieve the solution. If a solution cannot be found, or if Tasks determines that the costs are too high or a solution dependency cannot be met, then Tasks will adjust the requirements until an overall solution that meets the task can be found. Tasks makes this determination for all of the action components that it issues action requirements to.

In this case the action requirement can be met and the cost is acceptable; however, Target Engagement has identified a solution dependency to establish and maintain the position of the target to a required accuracy. Tasks therefore provides action requirements to Sensing to plan the necessary sensing activities and Sensor Data Interpretation to plan the sensor data interpretation activities. Sensing and Sensor Data Interpretation report their successful determination of a solution to Tasks with the associated costs and dependencies.

The solution dependencies from Sensing and Target Engagement include the need to get the air vehicle to the necessary locations for target detection and for release. Therefore, Tasks provides action requirements to Routes to provide a suitable route. Routes reports its successful determination of a solution to Tasks with the associated costs and dependencies.

Tasks determines that the requirements of the overall task can be met with acceptable costs.

C.2.9.1.3 Post-Conditions

- Weapon(s) have been selected for the engagement.
- The engagement is appropriately planned, including sequences of weapon deployment and associated dependencies.

C.2.9.2 Kinetic Attack

This IV illustrates an example of the components that interact to allow the system to execute an attack up to the point of weapon release. The attack is generated by an air-surface engagement task on a potentially mobile ground target, with a guided bomb under ownship self-designation.

It is assumed that:

- An agreed attack plan has been generated.
- Generating and securing authorisation for the attack plan has been previously carried out.
- The weapon is provided with targeting data (e.g. designator pulse repetition frequency codes and initial target coordinates) as well as airburst and impact settings as part of an approved attack plan.
- The weapon is not in a final armed state.
- The weapon has no routing capability.

The following considerations are excluded:

- Engaging with multiple weapon types or multiple weapons (either as a salvo release or as a separate repeated attack). This IV covers a single attack with a single weapon type only.
- All elements of the attack planning, as shown in the Plan For A/S Engagement IV, such as weapon type selection or LAR calculation during engagement planning. Only dynamic LAR refinement during engagement execution is covered.
- Details of the release process (e.g. involving Interlocks activating safety critical services for the resources needed to perform the release), are shown in the Releasing IV.
- Communications components, shown in the Data Transfer IV, as different weapons require different messaging systems.
- Other post-launch actions including activation of laser designation and collateral damage assessment.
- Refinement of predicted target movement.

C.2.9.2.1 Pre-Conditions

- Tasks holds the scheme for the previously planned attack and sets up coordination mechanisms between Target Engagement, Routes and Sensing, as well as a feedback channel between the components to enable execution of the attack.
- Authorisation has been given to execute the attack plan.

C.2.9.2.2 View

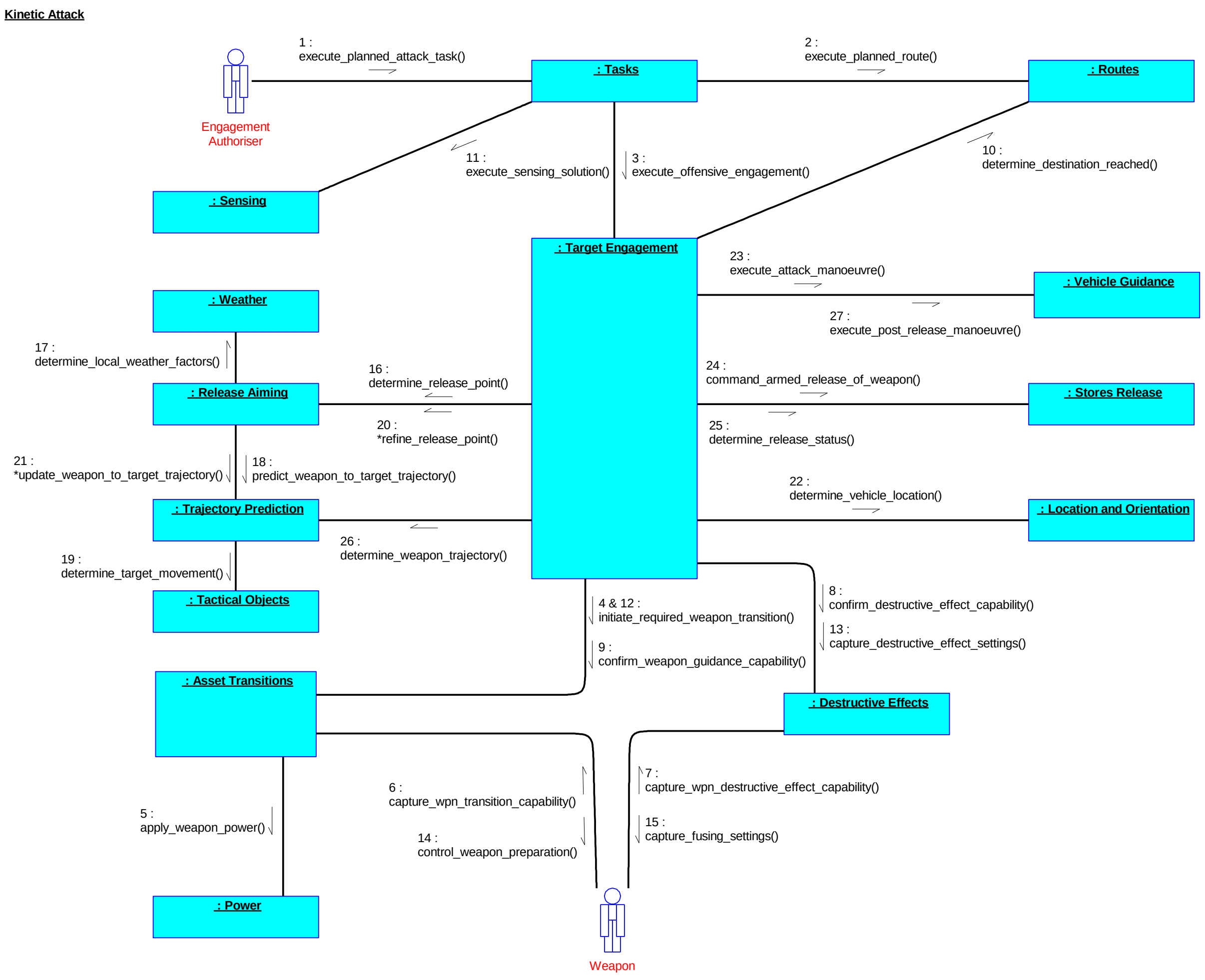

Kinetic Attack
Engagement Authoriser
1 : execute_planned_attack_task()
: Tasks
2 : execute_planned_route()
: Routes
: Sensing
11 : execute_sensing_solution()
3 : execute_offensive_engagement()
10 : determine_destination_reached()
: Target Engagement
23 : execute_attack_manoeuvre()
: Vehicle Guidance
27 : execute_post_release_manoeuvre()
: Weather
17 : determine_local_weather_factors()
16 : determine_release_point()
: Release Aiming
24 : command_armed_release_of_weapon()
: Stores Release
20 : *refine_release_point()
25 : determine_release_status()
21 : *update_weapon_to_target_trajectory()
18 : predict_weapon_to_target_trajectory()
22 : determine_vehicle_location()
: Trajectory Prediction
: Location and Orientation
26 : determine_weapon_trajectory()
19 : determine_target_movement()
: Tactical Objects
4 & 12 : initiate_required_weapon_transition()
8 : confirm_destructive_effect_capability()
13 : capture_destructive_effect_settings()
9 : confirm_weapon_guidance_capability()
: Asset Transitions
: Destructive Effects
7 : capture_wpn_destructive_effect_capability()
6 : capture_wpn_transition_capability()
5 : apply_weapon_power()
15 : capture_fusing_settings()
14 : control_weapon_preparation()
: Power
Weapon

Figure 200: Kinetic Attack IV

Tasks receives a request from the Engagement Authoriser to initiate a previously planned attack. Tasks uses the scheme for the previously planned solution as the basis of action requirements which are issued to the components containing the attack plan, with details consisting of: a weapons use and designation plan (held by Target Engagement), routes to acquire the target and to the release point (held by Routes) and sensing of the target (held by Sensing). Routes and Target Engagement are instructed to begin executing their parts of the attack plan.

Asset Transitions requests Power to apply power to the Weapon equipment, which in turn performs its own PBIT. Asset Transitions and Destructive Effects capture capability information from the BIT results of the Weapon equipment, which in turn supports Target Engagement in confirming the capability of the Weapon.

Routes executes the planned route and reports when the initiation point has been reached; Sensing executes actions that enable the detection of the target. Once the vehicle is correctly positioned, Target Engagement initiates preparation of the Weapon for release and confirms the destructive effect settings (e.g. impact fusing). Asset Transitions then transitions the Weapon to the correct state for a release (e.g. battery activation). Destructive Effects transfers the required fusing settings to the Weapon.

Release Aiming determines a planned release point, and refines this during manoeuvring by requesting Trajectory Prediction to continuously update the predicted weapon trajectory required for the weapon to reach the target. Information about weather and the target object is used to inform the aiming and trajectory solutions. If required to support the attack, Target Engagement will provide direct demands to Vehicle Guidance supported by Location and Orientation monitoring the vehicle position (for example to 'loft' a weapon in an extended-range attack).

Target Engagement initiates the operational release of the selected weapon by requesting Stores Release to perform the release in accordance with the release point defined by Release Aiming. The weapon is released in line with the Releasing IV.

The Exploiting Platform will need to provide weapon guidance at the appropriate phase of the weapon's flight (which is not part of this IV). To do this, the Exploiting Platform must be manoeuvred appropriately. Target Engagement requests Trajectory Prediction to determine the weapon flight path to allow the support manoeuvres to be determined. Target Engagement requests Vehicle Guidance to commence the support manoeuvres.

C.2.9.2.3 Post-Conditions

- A fully prepared and armed weapon has been safely released from the vehicle.
- The need for any post-release manoeuvres (either to maintain safe separation or to provide post release support) has been determined.

C.2.9.2.4 Actors

Engagement Authoriser

The authorised operator that authorises the engagement.

Weapon

The weapon chosen for engagement when generating the engagement plan.

C.2.10 Survivability Views

This section contains IVs relating to the survival capability of a system and how it is used.

The Figure 201: Survivability Views UC Diagram shows all the use cases within the Survivability Views and any actors that interact with them.

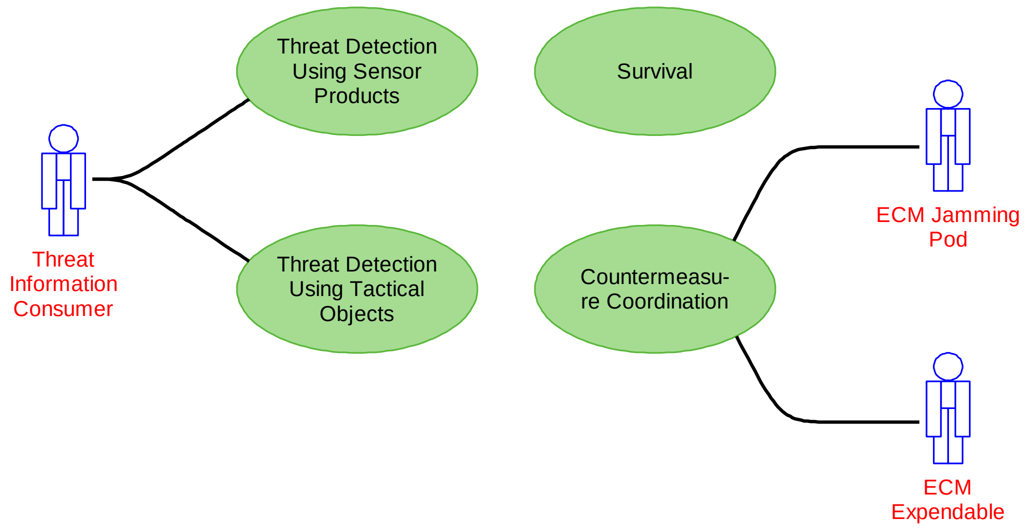

Threat Detection Using Sensor Products
Survival
Threat Information Consumer
ECM Jamming Pod
Threat Detection Using Tactical Objects
Countermeasu-re Coordination
ECM Expendable

Figure 201: Survivability Views UC Diagram

C.2.10.1 Survival

This IV illustrates how to determine, select and trigger a pre-planned mitigation strategy in response to a specific threat.

In order to focus the scope of the IV, the context has been limited to initially pre-planning a survival mitigation strategy based on knowledge of an expected type of threat, threat behaviour and threat condition (i.e. identifying a worst case scenario where a missile is launched towards the air vehicle). Then implementing the pre-planned survival strategy using the Action layer when the actual event occurs. This IV illustrates an example of how the Task and Action layers of the Control Architecture, with support from the tactical service based components, can coordinate together to achieve a survival goal.

It is assumed that:

- The engagement authority has pre-authorised survival tactics for use against a particular threat.

The following related areas of functionality are excluded:

- Determining situation awareness requirements and planning the actual collection of tactical information to support threat identification or behaviour assessment.
- The management of spectrum interoperability, mission constraints or mission re-planning changes as part of a mitigation strategy in response to a threat.
- Obtaining authorisation for the Countermeasures implementation scheme.

C.2.10.1.1 Pre-Conditions

- Tactical Objects holds a list of expected tactical objects (e.g. the SAM system types, their locations, and the missile types associated with them) found within the planned area of operation.
- All necessary survival tasking parameters (e.g. constraints, budgets, and criteria) are known and predefined within Tasks.
- The autonomy level for Countermeasures enactment has been defined.

C.2.10.1.2 View

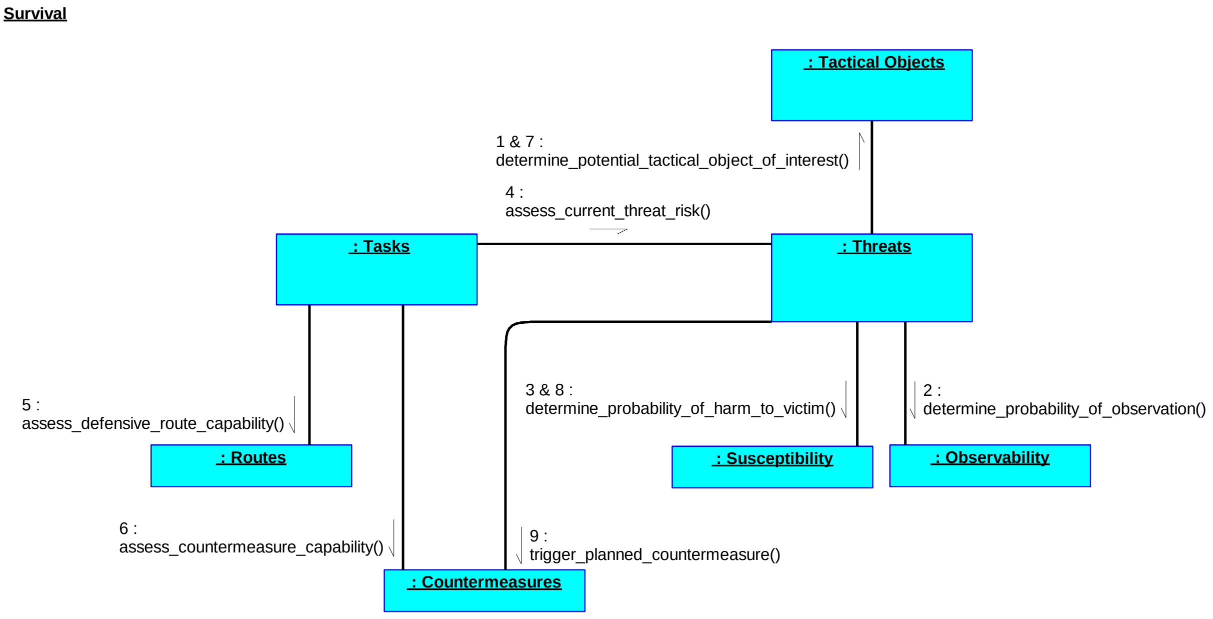

Survival
: Tactical Objects
1 & 7 : determine_potential_tactical_object_of_interest()
4 : assess_current_threat_risk()
: Tasks
: Threats
5 : assess_defensive_route_capability()
3 & 8 : determine_probability_of_harm_to_victim()
2 : determine_probability_of_observation()
: Routes
: Susceptibility
: Observability
6 : assess_countermeasure_capability()
9 : trigger_planned_countermeasure()
: Countermeasures

Figure 202: Survival IV

Threats identifies from Tactical Objects, Observability and Susceptibility that there is a high probability that a potential threat for a given threatening condition is likely to cause significant harm to the mission. Tactical Objects identifies the SAM system and its potential behaviours (i.e. search, tracking and locked-on, etc.). Observability determines that the SAM system radar's probability of observing the air vehicle is high. Susceptibility determines the SAM system has a high probability of harming the air vehicle if the air vehicle is observed. Threats provides the details of the threat and threatening conditions to Tasks.

Tasks determines a mitigation implementation scheme where the only threat mitigation capabilities available are routing or countermeasures and determines this would require an immediate mitigation response. Routes assesses whether it can determine an achievable route that could mitigate the threat. Countermeasures likewise determines its capability. Tasks examines all the pre-authorised tactics and determines that a countermeasure response (e.g. defensive manoeuvres, jamming and releasing of flares) which enables the air vehicle to continue its existing task is preferable to a defensive routing response (i.e. fly outside the weapon engagement zone) and determines a countermeasure implementation scheme.

Tasks creates a single mitigation action containing the pre-authorised implementation scheme for Countermeasures. Tasks sets a feedback link between Countermeasures and Threats where Threats is responsible for reporting to Countermeasures when the specific threat exceeds the threat risk threshold. Countermeasures determines the required countermeasure strategy for enactment. Tasks requires Threats to notify when the threat has exceeded the threat risk threshold and when it returns below the threshold in order to monitor the progress of the task.

Tactical Objects reports a new SAM system has been detected and its behaviour (i.e. locked on to the air vehicle) to Threats. Threats performs a threat risk assessment with support from Susceptibility. Susceptibility determines the probability of the SAM system harming the air vehicle is high and the effect would result in significant damage to the air vehicle. Threats concludes the threat has exceeded the threat risk threshold and informs Countermeasures. Countermeasures determines the enactment criteria has been fulfilled and executes the pre-determined countermeasure strategy. Tactical Objects reports the SAM system is no longer locked on. Threats re-assesses the threat risk and deems the threat has been reduced to an acceptable threat risk threshold. Threats passes the updated threat risk to Countermeasures. Countermeasures stops executing the mitigation plan.

C.2.10.1.3 Post-Conditions

- The survival mitigation response has successfully reduced the threat risk of the threat to an acceptable level.

C.2.10.2 Threat Detection

C.2.10.2.1 Threat Detection Using Sensor Products

This IV demonstrates how a threat can be identified using sensor data. This type of threat identification may be used in situations where a threat is expected to be present but Tactical Objects doesn't identify any relevant objects. Threats utilises Sensor Products to provide processed sensor information to inform the threat situation for a region of interest.

No assumptions are made for this IV.

The following related areas of functionality are excluded from this IV:

- Coordinating threat detection with other ongoing activities.
- The planning and coordination of an intelligence gathering task.
- The action coordination and execution of the tactical sensing solution.
- The determination and classification of an object.
- The planning and execution of threat mitigation.
- The countermeasure execution to a threat.

C.2.10.2.1.1 Pre-Conditions

- A threat identification library has been loaded.
- Threats has been tasked to assess for threats in a region of the battlespace.

C.2.10.2.1.2 View

Threat Detection Using Sensor Products

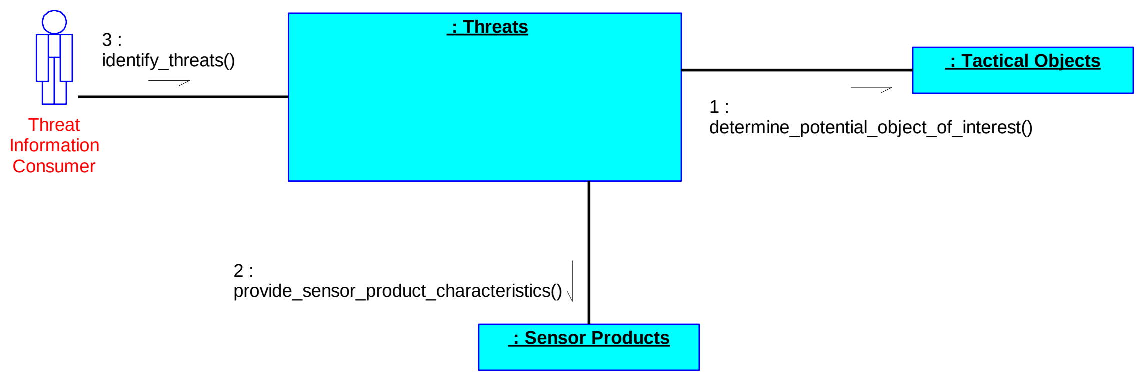

: Threats
: Tactical Objects
: Sensor Products
Threat Information Consumer
3 : identify_threats()
1 : determine_potential_object_of_interest()
2 : provide_sensor_product_characteristics()

Figure 203: Threat Detection Using Sensor Products IV

Threats requests battlespace entity information from Tactical Objects against threat criteria, this results in a situation where a threat is expected to be present but Tactical Objects doesn't identify any objects within the battlespace region of interest.

Threats requests Sensor Products to provide processed sensor data for the battlespace region of interest. Sensor Products provides the processed sensor data and Threats compares the sensor product characteristics against its internal threat library to identify a possible match against a potential threat. Threats identifies a match classifying it as a credible threat and reports the threat to the Threat Information Consumer.

C.2.10.2.1.3 Post-Conditions

- The Threat Information Consumer is aware of the threat against the air vehicle.

C.2.10.2.1.4 Actors

Threat Information Consumer

An entity which has an interest in actual threats which are required in order to fulfil its own responsibilities or required to make decisions such as an authorised operator or the Tasks component.

C.2.10.2.2 Threat Detection Using Tactical Objects

This IV demonstrates how a reported tactical object is identified as a threat to the ownship air vehicle. In this case, the scenario focuses on performing a threat identification and risk assessment on a newly observed tactical system, which comprises of a SAM early warning radar, SAM missile system and SAM fire control radar. As part of reporting the tactical system as a threat to the air vehicle, a threat assessment is performed to establish the potential of the SAM missile system (including its early warning radar, fire control radar and missiles) to engage the air vehicle based on the current planned route.

It is assumed that:

- Sensing has been configured during the pre-mission planning phase to prioritise sensing activities within frequency spectrums where more threats are expected to exist, to optimise tactical object identification.
- Observability and Susceptibility are already aware of the capabilities of any known threat types.
- A feedback loop has been established as part of the threat detection activity between Sensing and the relevant components to improve the sensing solution.

The following related areas of functionality are excluded from this IV:

- Coordinating threat detection with other ongoing activities.
- The planning and coordination of an intelligence gathering task.
- The action coordination and execution of the tactical sensing solution.
- The determination and classification of an object.
- The planning and execution of threat mitigation.
- The countermeasure execution to a threat.

C.2.10.2.2.1 Pre-Conditions

- A new tactical object has been added to the existing tactical object list.

C.2.10.2.2.2 View

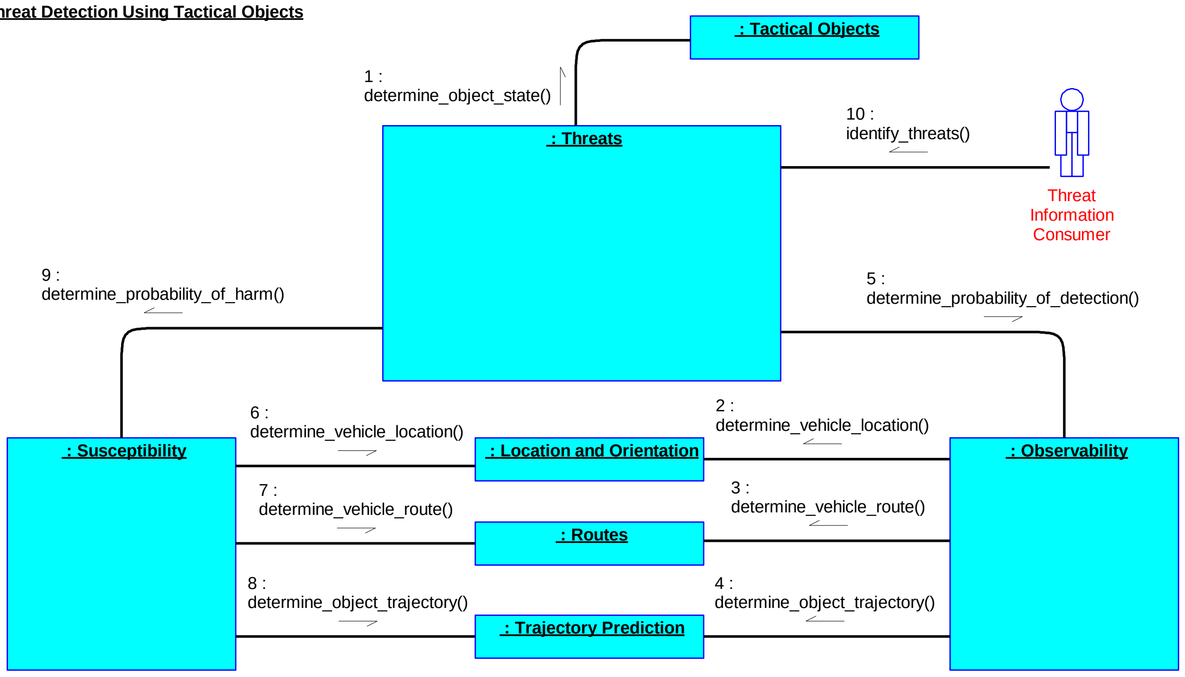

Threat Detection Using Tactical Objects
: Tactical Objects
1 : determine_object_state()
: Threats
10 : identify_threats()
Threat Information Consumer
9 : determine_probability_of_harm()
5 : determine_probability_of_detection()
6 : determine_vehicle_location()
2 : determine_vehicle_location()
: Susceptibility
: Location and Orientation
: Observability
7 : determine_vehicle_route()
3 : determine_vehicle_route()
: Routes
8 : determine_object_trajectory()
4 : determine_object_trajectory()
: Trajectory Prediction

Figure 204: Threat Detection Using Tactical Objects IV

Tactical Objects provides a new tactical object to Threats which includes information on the object type (i.e. SAM early warning radar), object state ("Search"), and object characteristics (i.e. "RF agile emitter", "Stable PRI", etc.). Tactical Objects also provides details of two other detected objects, one being a SAM missile system with an additional associated weapon, and the other being a tracking capability (SAM fire control radar). Tactical Objects identifies that these are three individual objects that form part of the same tactical system. Threats performs an initial threat assessment against the new tactical objects and determines that they constitute a potential threat.

Once the potential threat has been identified, Threats requests Observability to determine the probability of the air vehicle being detected. Observability needs to understand whether the potential threat is within the boundaries of the vehicles planned route. Observability requests the current vehicle location from Location and Orientation, the current route from Routes and the predicted trajectory of the tactical objects from Trajectory Prediction. Using this information, Observability reports to Threats that the probability of being detected is high.

Threats needs to understand whether the potential threat can cause significant harm to the air vehicle if the air vehicle continues its current planned route. Threats requests Susceptibility to determine the probability of the air vehicle being harmed. Using relevant information from Location and Orientation, Routes and Trajectory Prediction, Susceptibility determines the probability of harm to the air vehicle and reports it to Threats.

Threats processes all of the information through the threat assessment algorithm and determines that the sensed tactical system is a credible threat to the air vehicle if the air vehicle continues its planned route. Threats reports the credible threat to the Threat Information Consumer.

C.2.10.2.2.3 Post-Conditions

- The Threat Information Consumer is aware of the threat against the air vehicle.

C.2.10.2.2.4 Actors

Threat Information Consumer

An entity which has an interest in actual threats which are required in order to fulfil its own responsibilities or required to make decisions such as an authorised operator or the Tasks component.

C.2.10.3 Countermeasure Coordination

This IV describes how to coordinate and execute a planned countermeasure strategy in response to a specific threat.

In order to focus the scope of the IV, the context has been limited to coordinating and executing an existing countermeasure plan in response to a specific threat. The aim of the scenario is to disrupt a threat radar's "tracking" signal which is used to track the air vehicle. The countermeasure response involves initially executing a defensive manoeuvre and then performing an ECM program. The last countermeasure response is to release chaff which causes significant tracking errors to the threat radar and completes the disruption. The scenario will determine the success of the countermeasure response when observing a change in the threat's behaviour (i.e. changes from "Tracking to Search").

This IV illustrates an example of how the Action and Resource layers of the Control Architecture with support from the tactical service based components can adaptively coordinate together to achieve a countermeasure plan.

It is assumed that:

- Countermeasure tactics are preauthorised against particular threats.
- The survival tactics specify the type of defensive manoeuvre including the associated trigger point to execute for a predefined threat.
- The ECM program parameters will be predefined for a specific threat.
- The threat has the ability to change its waveform transmissions in attempt to mitigate interference from ECM jamming.
- The adaptive ECM waveform generation is considered as an inherent capability of the ECM Jamming Pod however the ECM programmable parameters (e.g. beam direction and time to completion) will be input requirements to the ECM Jamming Pod.
- There are no resource limitations with consumable expendables for survivability purposes.
- There are no operational limits on the countermeasure response to a threat.

The following related areas of functionality are excluded:

- Determining situation awareness requirements and planning the actual collection of tactical information to support threat identification or behaviour assessment.
- The identification and selection of a survival mitigation strategy for a specific threat.
- The management of spectrum interoperability.
- The details of releasing ECM expendables.
- The interlocks required to support the enactment.
- The execution of a vehicle manoeuvre.
- Determination of atmospheric conditions.
- Predicting the trajectory of ECM expendables post-release.

C.2.10.3.1 Pre-Conditions

- A survival plan has been agreed.
- Countermeasures has been preauthorised to enact the countermeasure response to a particular type of threat.
- A feedback link has been established between Countermeasures and Threats.
- A threat is known to the system and is being tracked.

C.2.10.3.2 View

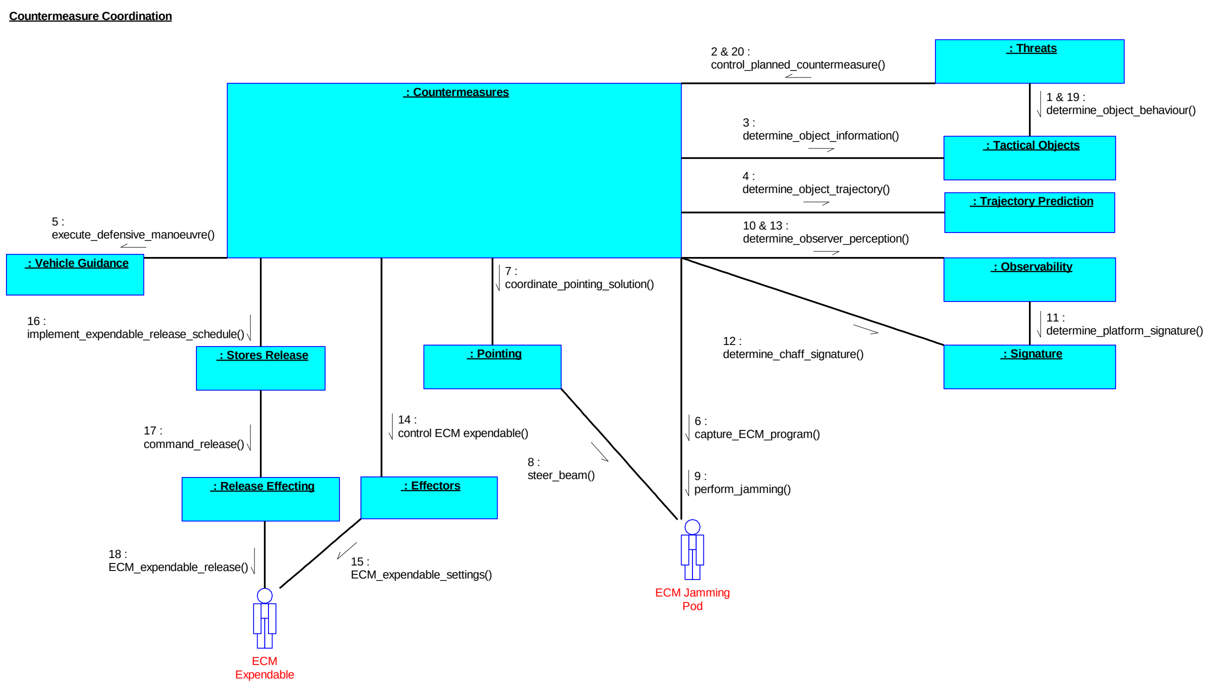

Countermeasure Coordination
: Countermeasures
: Threats
: Tactical Objects
: Trajectory Prediction
: Observability
: Signature
: Vehicle Guidance
: Stores Release
: Pointing
: Release Effecting
: Effectors
ECM Jamming Pod
ECM Expendable
1 & 19 : determine_object_behaviour()
2 & 20 : control_planned_countermeasure()
3 : determine_object_information()
4 : determine_object_trajectory()
5 : execute_defensive_manoeuvre()
6 : capture_ECM_program()
7 : coordinate_pointing_solution()
8 : steer_beam()
9 : perform_jamming()
10 & 13 : determine_observer_perception()
11 : determine_platform_signature()
12 : determine_chaff_signature()
14 : control ECM expendable()
15 : ECM_expendable_settings()
16 : implement_expendable_release_schedule()
17 : command_release()
18 : ECM_expendable_release()

Figure 205: Countermeasure Coordination IV

Tactical Objects determines the object is in "tracking mode". Threats reports to Countermeasures when a threat has exceeded the threat risk threshold. Countermeasures directs Tactical Objects and Trajectory Prediction to provide additional information about the threat which needs a countermeasure response.

In accordance with the agreed survival plan, Countermeasures begins by tasking Vehicle Guidance with a specific defensive manoeuvre to execute.

Countermeasures provides the ECM Jamming Pod with a predefined ECM program based on the information provided by Tactical Objects, Trajectory Prediction and countermeasure enactment criteria. In addition, Countermeasures specifies to the ECM Jamming Pod the ECM program run time. Countermeasures tasks Pointing to position the ECM Jamming Pod towards the intended target.

The execution of the ECM program is performed by the ECM Jamming Pod. Pointing commands the ECM Jamming Pod to direct the electronic beam towards the intended target. The ECM Jamming Pod steers the direction of the electronic beam in order to jam the main lobe of the intended target. In addition, the ECM Jamming Pod translates the ECM program into jamming transmissions. The ECM Jamming Pod adapts its jamming waveform in response to the agile threat.

Concurrently, Countermeasures actions Observability to estimate how the object currently perceives the air vehicle. To do this, Observability directs Signature to calculate the current radar cross section of the air vehicle. Observability then produces an observability assessment based on the calculated radar cross section of the air vehicle and reports this back to Countermeasures. Countermeasures uses this assessment to determine a chaff release countermeasure solution. Countermeasures directs Signature to calculate the radar cross section of the chaff packages that are options for a chaff countermeasures solution. Countermeasures actions Observability to estimate how the object would perceive the air vehicle based on the release of suggested chaff package options. Countermeasures determines the best chaff program to use based on the observability values and current survivability rules.

Countermeasures determines the chaff program properties (e.g. bloom rate, decay rate, dispense rate, polarisation and chaff length), with any internal settings of the chaff program being controlled via Effectors. Countermeasures provides the chaff countermeasure program and enactment criteria to Stores Release. Stores Release commands Release Effecting to execute the dispensing of the chaff.

Tactical Objects determines the object has returned to "search mode". Threats performs a threat risk assessment and determines the threat risk threshold has decreased. Countermeasures stops the countermeasure execution.

C.2.10.3.3 Post-Conditions

- The countermeasure solution has successfully reduced the threat risk of the threat to an acceptable level.

C.2.10.3.4 Actors

ECM Expendable

A consumable item that can be released from the vehicle for defensive or survivability purposes.

ECM Jamming Pod

A physical pod that is capable of receiving, synthesising and effecting within the RF portion of the EM spectrum.

C.2.11 Contingency Views

This section contains IVs relating to the contingency management capability of a system and how it is used.

The Figure 206: Contingency Views UC Diagram shows all the use cases within the Contingency Views.

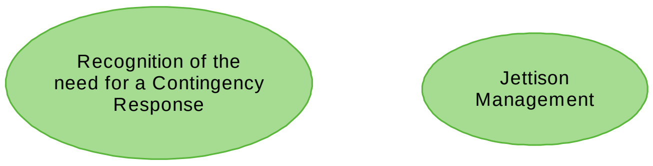


Figure 206: Contingency Views UC Diagram

C.2.11.1 Recognition of the need for a Contingency Response

The purpose of this IV is to illustrate how an abnormal rate of fuel depletion is identified and assessed as requiring a contingency response.

The following considerations are excluded:

- The full range of system inputs that may be used to identify the anomaly and determine its root cause.
- Failure handling (i.e. attempting to rectify the problem).
- Communication with the authorised operator.
- The determination of the contingency task.

C.2.11.1.1 Pre-Conditions

- The aircraft is airborne.
- Fuel is being depleted at a greater than expected rate.

C.2.11.1.2 View

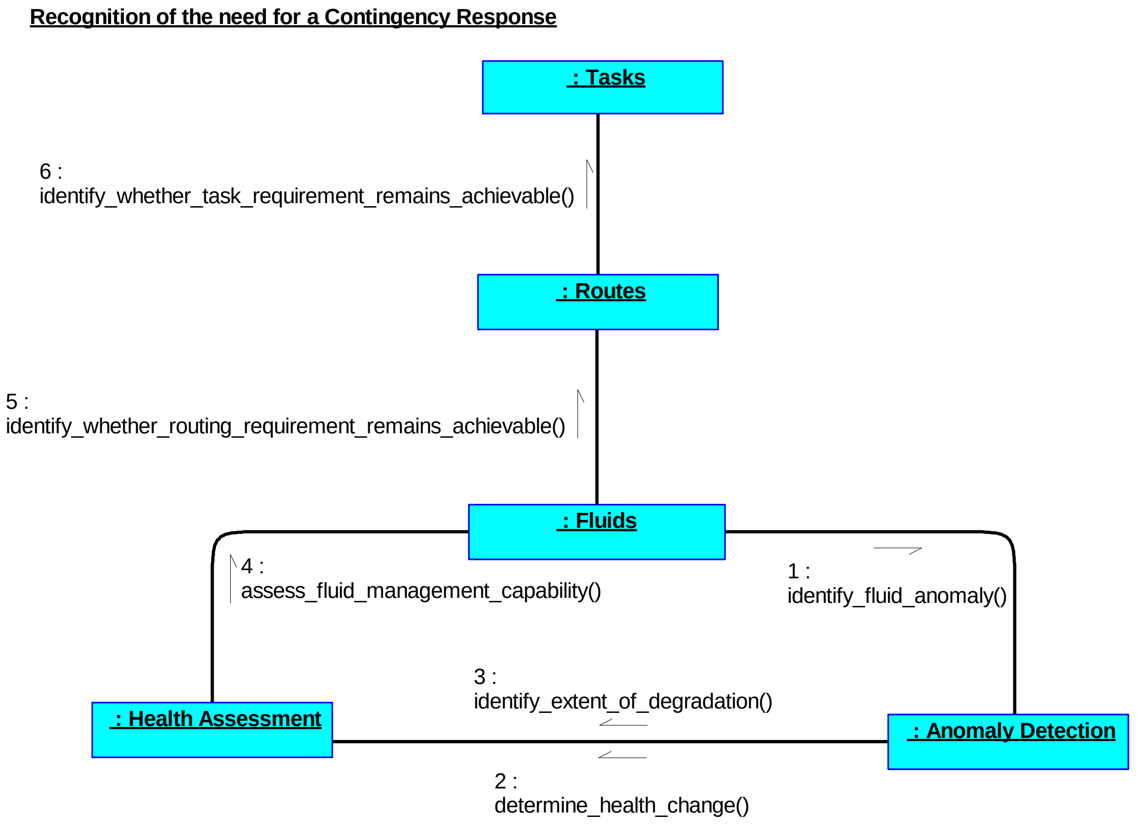

Recognition of the need for a Contingency Response
: Tasks
6 : identify_whether_task_requirement_remains_achievable()
: Routes
5 : identify_whether_routing_requirement_remains_achievable()
: Fluids
4 : assess_fluid_management_capability()
1 : identify_fluid_anomaly()
3 : identify_extent_of_degradation()
: Health Assessment
: Anomaly Detection
2 : determine_health_change()

Figure 207: Recognition of the need for a Contingency Response IV

Anomaly Detection identifies that an anomaly is present as a result of an abnormal rate of fuel depletion. Health Assessment identifies the cause of the anomaly as a damaged fuel tank and determines the extent of the damage. Fluids reassesses its capability to store fuel and reports an inability to meet its managed resource allocation to Routes. Routes determines that it can no longer meet the current positioning requirements and reports a change in routing capability to Tasks. Tasks determines that it can no longer meet the requirements of the transit task, and determines the change in transit capability. Tasks identifies a new contingency task to mitigate the reduction in transit capability.

C.2.11.1.3 Post-Conditions

- A new contingency task has been identified.

C.2.11.2 Jettison Management

This IV illustrates the determination and subsequent enactment of a jettison requirement as a method to achieve a reduction in the air vehicle's weight. In this instance, multiple independent elements require jettison, necessitating the need for Jettison to coordinate the overall jettison implementation.

While this IV is only scoped to the reduction of weight being the triggering requirement for jettison, other requirements that require jettison as a mitigation strategy also exist, such as the need to make the air vehicle more aerodynamically efficient.

This IV also shows the routing of the air vehicle to an approved jettison area to allow the jettison to take place.

It is assumed that:

- Jettison is limited only to stores fitted to the air vehicle (e.g. weapons, expendable pods and fuel tanks) and the dumping of fuel directly.
- There is at least 1 store with sensitive data within the jettison package that requires sanitisation prior to jettison.
- The release occurs successfully after jettison is initiated.

The following considerations are excluded:

- Manual triggering of jettison.
- Details of flying the vehicle to a jettison location.
- Details of the physical release of store(s) and fuel from the air vehicle.
- Priming of any fitted weapons with jettison specific data.
- The process of obtaining authorisation for jettison.

C.2.11.2.1 Pre-Conditions

- The air vehicle is fitted with jettisonable stores and fuel.
- Some stores are behind doors on the air vehicle.
- The Jettison component has knowledge of allowable jettison locations.

C.2.11.2.2 View

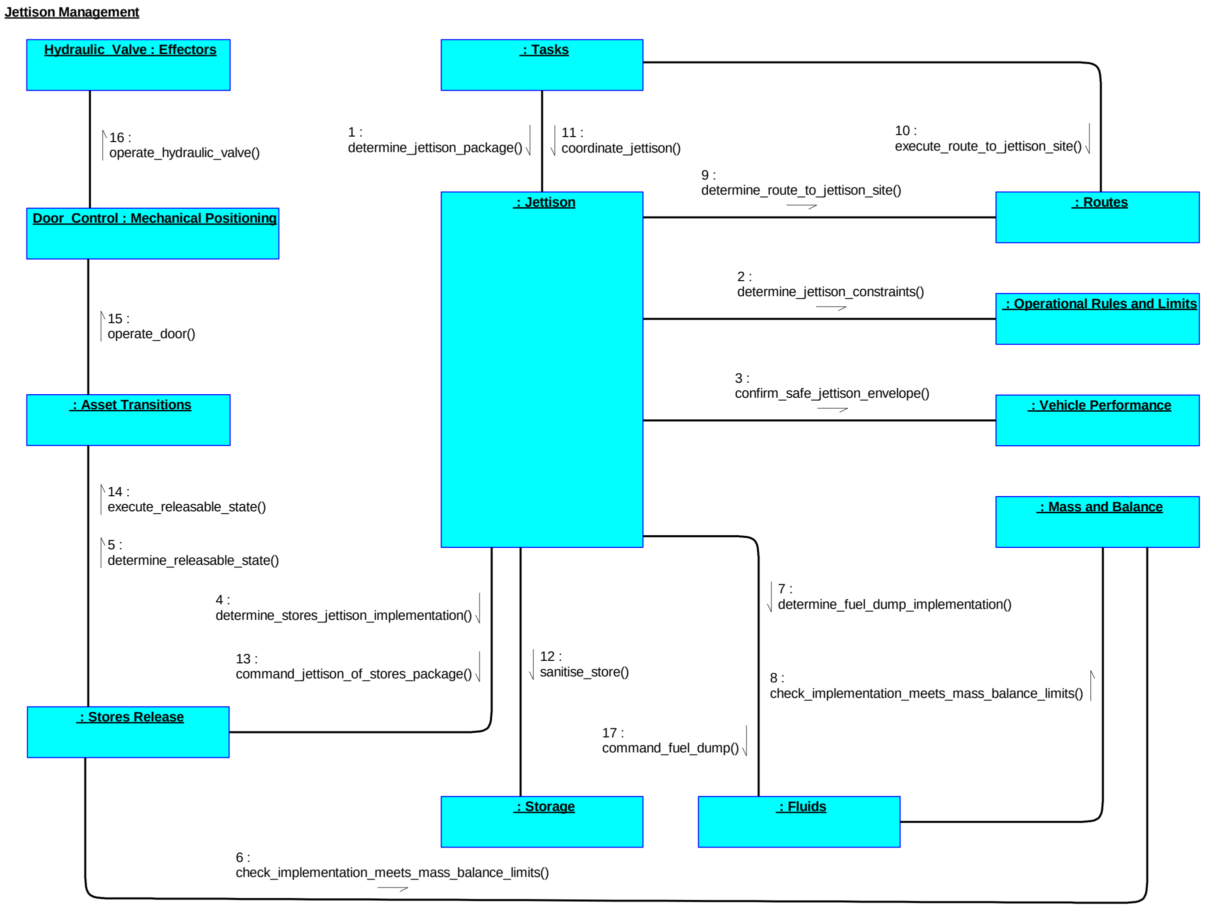

Jettison Management
Hydraulic Valve : Effectors
: Tasks
16 : operate_hydraulic_valve()
1 : determine_jettison_package()
11 : coordinate_jettison()
10 : execute_route_to_jettison_site()
9 : determine_route_to_jettison_site()
Door Control : Mechanical Positioning
: Jettison
: Routes
2 : determine_jettison_constraints()
: Operational Rules and Limits
15 : operate_door()
3 : confirm_safe_jettison_envelope()
: Asset Transitions
: Vehicle Performance
14 : execute_releasable_state()
: Mass and Balance
5 : determine_releasable_state()
7 : determine_fuel_dump_implementation()
4 : determine_stores_jettison_implementation()
13 : command_jettison_of_stores_package()
12 : sanitise_store()
8 : check_implementation_meets_mass_balance_limits()
: Stores Release
17 : command_fuel_dump()
: Storage
: Fluids
6 : check_implementation_meets_mass_balance_limits()

Figure 208: Jettison Management IV

Tasks determines there is a requirement to make the vehicle lighter to achieve a task, this request is sent to Jettison. Jettison determines a jettison package that consists of stores and fuel that will achieve the desired reduction of weight. As part of the solution, Jettison ensures that the jettison package conforms to limits provided by Operational Rules and Limits and Vehicle Performance.

Jettison requests that Stores Release determines a stores release solution for the jettison package. Stores Release determines that the jettison package is an allowable release package and checks with Mass and Balance that its solution is acceptable (no other stores need to be released in addition for mass and balance reasons) and determines a release sequence to release them. Asset Transitions is instructed by Stores Release to determine an infrastructure state that will make the jettison package releasable (i.e. opening doors before release).

Jettison requests that Fluids determines a fuel dump solution and Fluids checks with Mass and Balance that its solution is acceptable.

Jettison determines that the jettison is feasible, but that the air vehicle must first be flown to an approved jettison location. Jettison asks Routes to determine the route to the jettison location, and Tasks begins executing the jettison task by instructing Routes to fly to the desired location.

Once the desired location has been reached, Tasks commands the execution of the planned jettison solution. Jettison requests Storage to sanitise any stores in the jettison package that contain sensitive data. Jettison then commands Stores Release to execute jettison of the stores. Asset Transitions is instructed by Stores Release to execute an infrastructure state change that will make the jettison package releasable. This is executed via Door_Control: Mechanical Positioning and Hydraulic Valve: Effectors (as the doors are operated by hydraulic pressure).

After the store has been jettisoned, Jettison commands Fluids to perform the fuel dump.

C.2.11.2.3 Post-Conditions

- A coordinated jettison of a stores package and a fuel dump has been executed.

C.2.12 Resource Views

This section contains IVs relating to the physical resources of a system.

The Figure 209: Resource Views UC Diagram shows all the use cases within the Resource Views and any actors that interact with them.

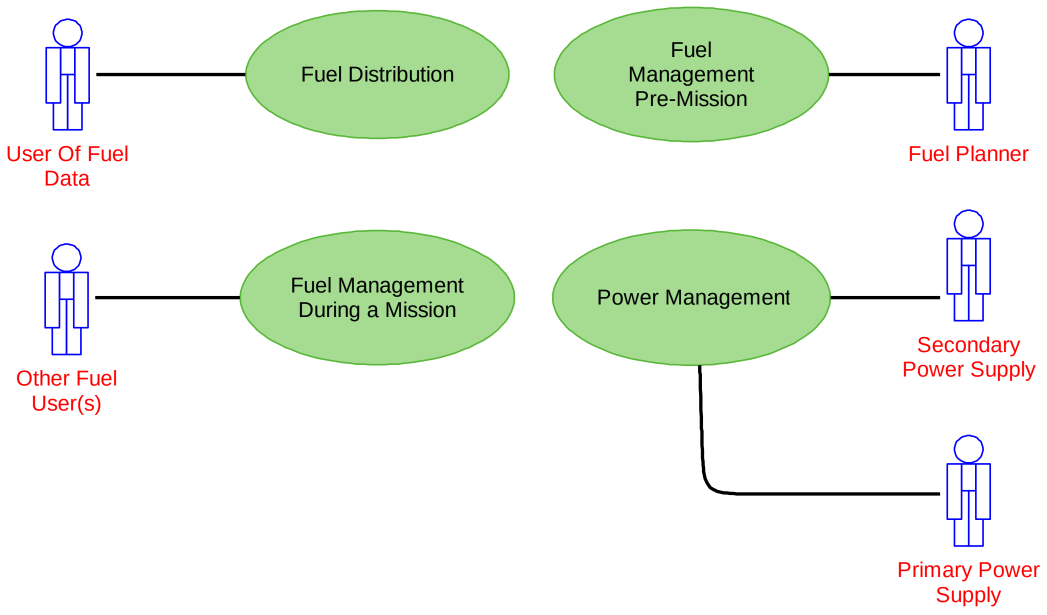


Figure 209: Resource Views UC Diagram

C.2.12.1 Fuel Distribution

This IV describes a specific example of fuel being made available to the engines of the air vehicle. The distribution of fuel within the tanks takes into account centre of gravity constraints provided by Mass and Balance.

It is assumed that:

- Fluids has knowledge of all vehicle fuel.
- Fuel is a liquid.

The following related areas of functionality are excluded:

- Health assessment / anomaly detection.
- Refuelling.
- Management of jettison of fuel / fuel tanks.
- HMI.
- Controlling the rate of fuel burn by the engines, in response to throttle demands.

C.2.12.1.1 Pre-Conditions

- System is initialised and running with all removable tanks accounted for by Inventory.

C.2.12.1.2 View

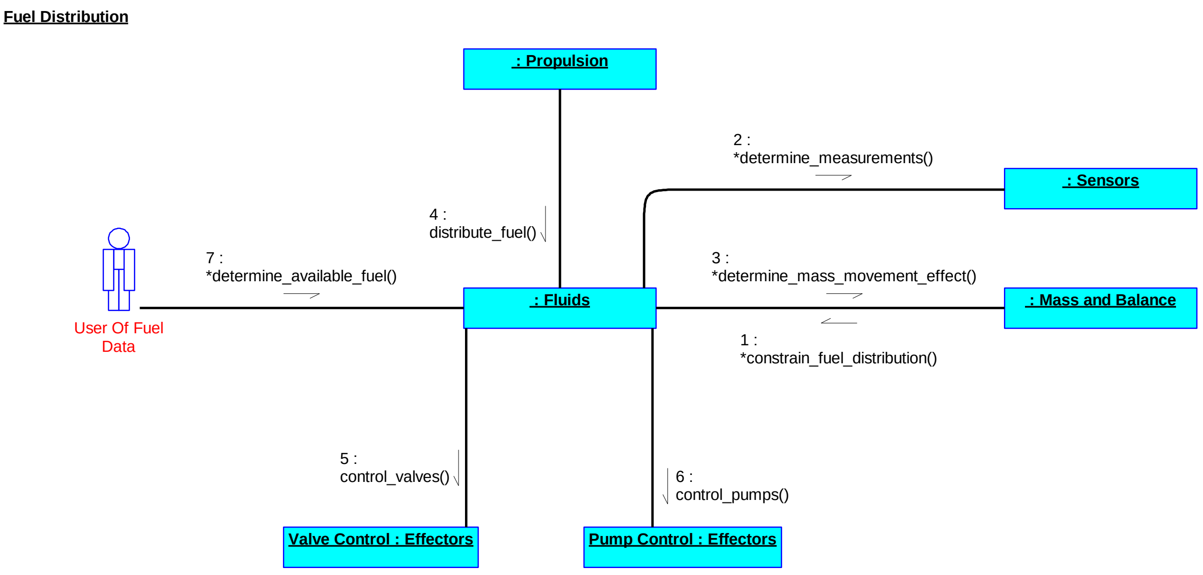


Figure 210: Fuel Distribution IV

Mass and Balance continuously provides Fluids with constraints regarding the distribution of fuel, to ensure that the centre of gravity remains within limits. Fluids determines from Sensors the raw measurements (e.g. capacity and temperature) so that Fluids can determine the quantity of fuel. Fluids reports to Mass and Balance how the fuel is distributed so that Mass and Balance can determine the effect that the distribution may have on the centre of gravity, mass and inertia. In response to a demand by Propulsion, Fluids makes fuel available to engine(s) using Valve_Control: Effectors and Pump_Control: Effectors (to enable fuel transfer and venting), subject to Mass and Balance constraints. Fluids provides fuel related information, such as the amount of available fuel, to a User Of Fuel Data.

C.2.12.1.3 Post-Conditions

- A fuel demand from Propulsion is met, and the centre of gravity remains within limits.

C.2.12.1.4 Actors

User Of Fuel Data

A user of the fuel data provided by Fluids. This may be a crew member observing the data using an HMI or another component within the architecture.

C.2.12.2 Fuel Management

C.2.12.2.1 Fuel Management Pre-Mission

This IV describes the high level planning of fuel requirements needed for a mission, prior to mission commencement. Routes determines the fuel costs in order to fulfil the mission requirements placed upon it by Tasks.

It is assumed that:

- Fluids has received no other unresolved demands for fuel allocation.

The following related areas of functionality are excluded:

- Contingency fuel planning.
- HMI.
- Mission data load.

C.2.12.2.1.1 Pre-Conditions

- A task exists that requires a plan to be generated.

C.2.12.2.1.2 View

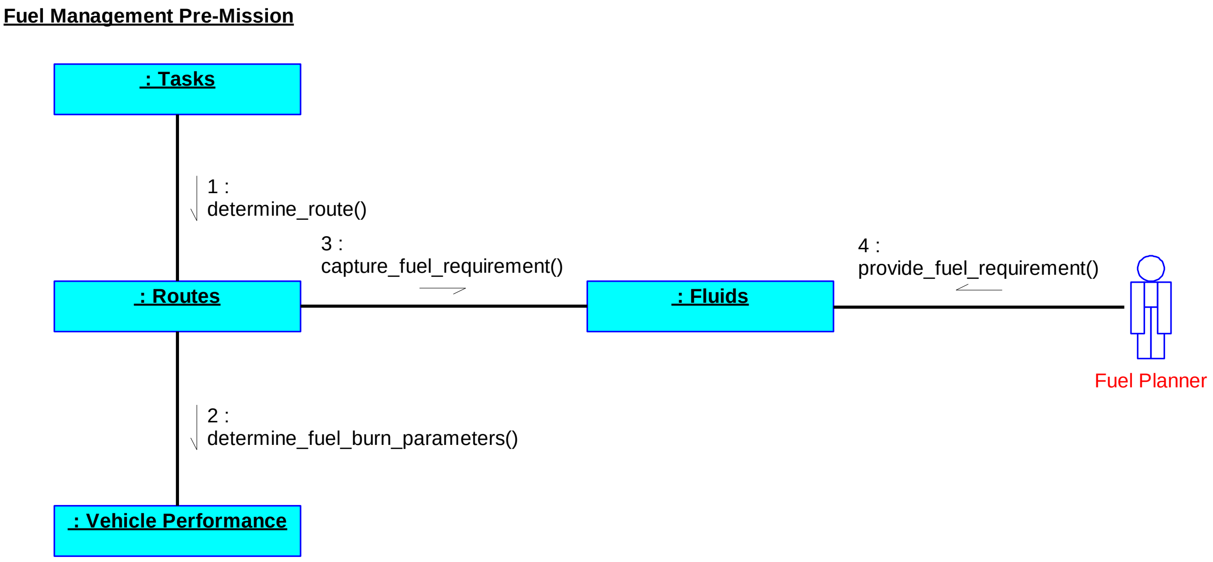

Fuel Management Pre-Mission
: Tasks
1 : determine_route()
: Routes
3 : capture_fuel_requirement()
: Fluids
4 : provide_fuel_requirement()
Fuel Planner
2 : determine_fuel_burn_parameters()
: Vehicle Performance

Figure 211: Fuel Management Pre-Mission IV

Tasks requests a route from Routes based upon requirements. Routes determines a route to the specified location and collates the fuel cost for this route from Vehicle Performance. Fluids then allocates the fuel based on requests placed upon it from Routes and provides the Fuel Planner with the fuel requirement.

C.2.12.2.1.3 Post-Conditions

- The Fuel Planner knows the fuel requirement.

C.2.12.2.1.4 Actors

Fuel Planner

The person responsible for creating a fuelling plan from the fuel requirements.

C.2.12.2.2 Fuel Management During a Mission

This IV describes how an air vehicle's fuel reserves are allocated and monitored during a mission. The vehicle is following a planned route, but is forced to perform a replan due to a change in routing requirements.

It is assumed that:

- The mission objectives are still achievable after a route replan due to sufficient fuel reserves.

The following related areas of functionality are excluded:

- Further detail of route generation, authorisation and progression.
- HMI.

C.2.12.2.2.1 Pre-Conditions

- The vehicle is in flight.
- Tasks has received a requirement to change route.

C.2.12.2.2.2 View

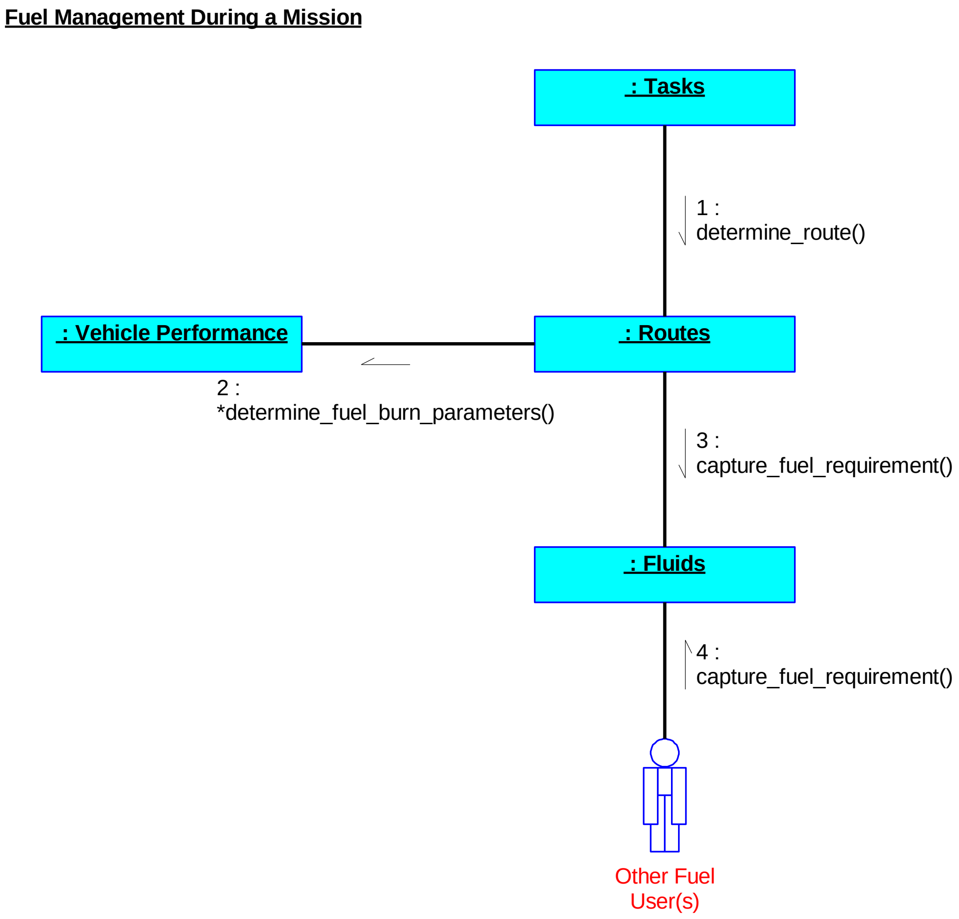

Fuel Management During a Mission
: Tasks
1 : determine_route()
: Vehicle Performance
: Routes
2 : *determine_fuel_burn_parameters()
3 : capture_fuel_requirement()
: Fluids
4 : capture_fuel_requirement()
Other Fuel User(s)

Figure 212: Fuel Management During a Mission IV

Routes continually updates the planned route fuel cost based on updates from Vehicle Performance. Concurrently, Fluids determines its contents and allocates the updated route fuel cost alongside updates from Other Fuel User(s), while also ensuring that there is sufficient unallocated fuel.

Tasks requests Routes to determine a suitable new route. Routes requests that Vehicle Performance calculate the fuel cost for the newly revised route. The fuel cost is sent to Routes. Fluids receives the request for fuel required for the new route along with the requests of other fuel users. Routes determines that in this scenario there is sufficient fuel for the route change.

The revised route fuel is then allocated and a request for route authorisation is ready to be sent.

C.2.12.2.2.3 Post-Conditions

- The route has been successfully updated.

C.2.12.2.2.4 Actors

Other Fuel User(s)

A planned user of fuel with a corresponding fuel cost previously calculated.

C.2.12.3 Power Management

This IV illustrates how electrical power management can be implemented for an Exploiting Platform, in this case an air vehicle. It is anticipated that, under normal operating conditions, the demand for electrical power will be adequately met by the supplies available, however there may be occasions when this is not true and the available supplies are restricted, e.g. following a failure or damage to the electrical systems (generators, busbars, etc.) resulting in a diminished power system capability. This IV describes such a situation when demand has to be managed in order not to exceed the available supply. It includes demands from a high powered sensor (including for cooling), engines in maximum reheat and the pre-launch conditioning requirements of a missile, but other demands could be added or substituted.

It is assumed that:

- The air vehicle is operating on its own electrical supply, with on board generators and battery reserves.
- Any required power management usage profiles, prioritisation policies and interruptibility strategies are loaded into the system.
- The available short range missile requires a dedicated start-up sequence before it can be used and also requires pre-launch conditioning of the missile sensor.
- Additional power required for flight-critical services (such as cooling the equipment in the avionics bay) is ring-fenced.
- The control of the cooling aspects of the scenario is limited to a simple enabling / disabling of power to the cooling systems. Hence the resource required by the Environmental Conditioning component in order to fulfil its actions is power, and the power demand is made by that component directly. The other cases in the scenario are different in that the resources required by the other Action Layer components are the associated hardware (i.e. engine, sensor or weapon), with power being a secondary resource enabling the capability of that hardware. Hence the power demands are made from the associated resource user components (Propulsion, Sensors or Asset Transitions) in those cases.

The following related areas of functionality are excluded:

- The specific chain of components which translate the solution created by Vehicle Guidance into specific demands on Propulsion.
- Intermediate steps between Tasks and Vehicle Guidance.
- The weapon selection chain beyond Target Engagement.

C.2.12.3.1 Pre-Conditions

- Whilst carrying out an air vehicle mission, tasking is received to intercept and engage a target as a priority.
- Continuous monitoring of system health has determined that there is damage to a generator, and so this generator should not be used. Therefore, the Power component has determined that the system must operate with reduced power availability.

C.2.12.3.2 View

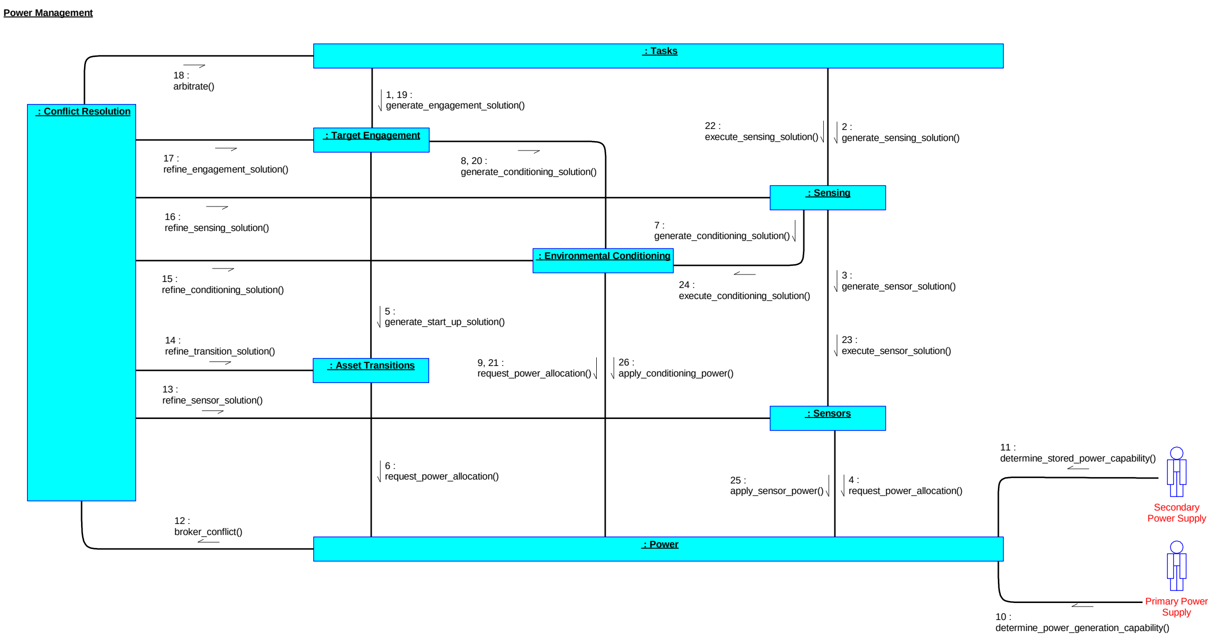

Power Management
: Tasks
: Conflict Resolution
: Target Engagement
: Sensing
: Environmental Conditioning
: Asset Transitions
: Sensors
: Power
18 : arbitrate()
1, 19 : generate_engagement_solution()
17 : refine_engagement_solution()
8, 20 : generate_conditioning_solution()
22 : execute_sensing_solution()
2 : generate_sensing_solution()
16 : refine_sensing_solution()
7 : generate_conditioning_solution()
15 : refine_conditioning_solution()
24 : execute_conditioning_solution()
3 : generate_sensor_solution()
5 : generate_start_up_solution()
14 : refine_transition_solution()
9, 21 : request_power_allocation()
26 : apply_conditioning_power()
23 : execute_sensor_solution()
13 : refine_sensor_solution()
6 : request_power_allocation()
25 : apply_sensor_power()
4 : request_power_allocation()
11 : determine_stored_power_capability()
Secondary Power Supply
Primary Power Supply
10 : determine_power_generation_capability()
12 : broker_conflict()

Figure 213: Power Management IV

Tasks provides initial tasking to:

- Generate a sensing solution to track the target;
- Generate an engagement solution for engaging the target.

As a result of this:

- Sensing determines the sensor capability required to execute the sensing solution and requests Sensors to determine how to provide that capability. Sensors determines that power is required and requests an allocation from Power.
- Target Engagement determines the weapon capability required to execute the engagement solution - in this case there is a dedicated start-up sequence required for the missile so Target Engagement requests Asset Transitions to generate the weapon start-up solution. Asset Transitions determines that power is required as part of the start-up sequence and requests an allocation from Power.
- Environmental Conditioning receives the requirements to cool both the sensor and the missile (from Sensing and Target Engagement respectively). It determines that power is required and requests an allocation from Power.

Power obtains the necessary information that allows it to determine the available supply of electrical energy and the stored battery reserves. Based on the power available for allocation, Power determines that the demand for power outstrips the available supply and, in this case, Conflict Resolution is informed of the conflict in the power allocation.

Conflict Resolution informs Environmental Conditioning, Asset Transitions and Sensors that their respective solutions need to be refined to meet the power requirements. Environmental Conditioning, Asset Transitions and Sensors analyse and refine their solutions to meet the power requirements.

Environmental Conditioning and Asset Transitions inform Conflict Resolution that they cannot refine their solutions to meet the requirements placed on them by Sensing and Target Engagement.

Conflict Resolution informs Sensing and Target Engagement of the need to refine their solutions. Both components determine that their solutions cannot be refined and inform Conflict Resolution of their inability to generate the requested solutions.

Conflict Resolution requests Tasks to arbitrate a solution. Tasks updates its task requirements such that sensing actions take precedence over the engagement of the target (i.e. the requirement for Target Engagement to generate an engagement solution at this time is removed). The process described above is repeated, but with a reduced power allocation being required by Environmental Conditioning. Power indicates that the reduced power demands can be allocated concurrently and hence Sensing informs Tasks that solutions can be generated. Tasks commands Sensing to execute its solutions and Sensing commands Environmental Conditioning to execute the updated conditioning solution (i.e. to cool the sensor only). Demands for the power allocations already agreed are made by Sensors and Environmental Conditioning.

C.2.12.3.3 Post-Conditions

- Missile conditioning is delayed, while precedence is given to intercept trajectory and tracking.

C.2.12.3.4 Actors

Secondary Power Supply

Auxiliary or backup electrical Power_Source (e.g. a battery).

Primary Power Supply

The primary Power_Source for electrical power (e.g. a generator within the propulsion system).

Appendix D: Deployment Guide

D.1 Introduction

This appendix outlines system and software design strategies that could be adopted when using the PRA to develop air systems, and how artefacts within the PRA and PYRAMID Technical Standard Guidance document could be utilised by an Exploiting Programme to achieve the intended PYRAMID benefits. It identifies at what stage in the design and development lifecycle the different artefacts are most useful and includes:

- What stages are normally involved in the design and development of a PYRAMID deployment;
- What support is provided by the contents of the PYRAMID Technical Standard, Ref. [1] and PYRAMID Technical Standard Guidance document, to assist engineers with these stages of development;
- What should be considered in order to develop a PYRAMID deployment that allows the PYRAMID benefits to be realised.
- What potential impacts an Exploiting Programme may experience when updated versions of the PRA are issued.

A deployment is defined in the glossary (Appendix F: Glossary) as “a set of hardware and software elements forming a system (or part thereof) that satisfy the overall system requirements.” As the PRA is a reference architecture for the software aspects of functionality of an air system, the term ‘deployment’ is used in this appendix to refer to a software-based system implementation based on the PRA.

The scope of this appendix is therefore limited to providing guidance regarding the approach to developing the application-level software for the parts of the Exploiting Platform that are intended to be developed based on the PRA. In the example shown in Figure 214: Example Scope of Deployment within an Exploiting Platform this is shown as an air vehicle sub-system; however, the scope can range from a single component to all of the application software for an Exploiting Platform.

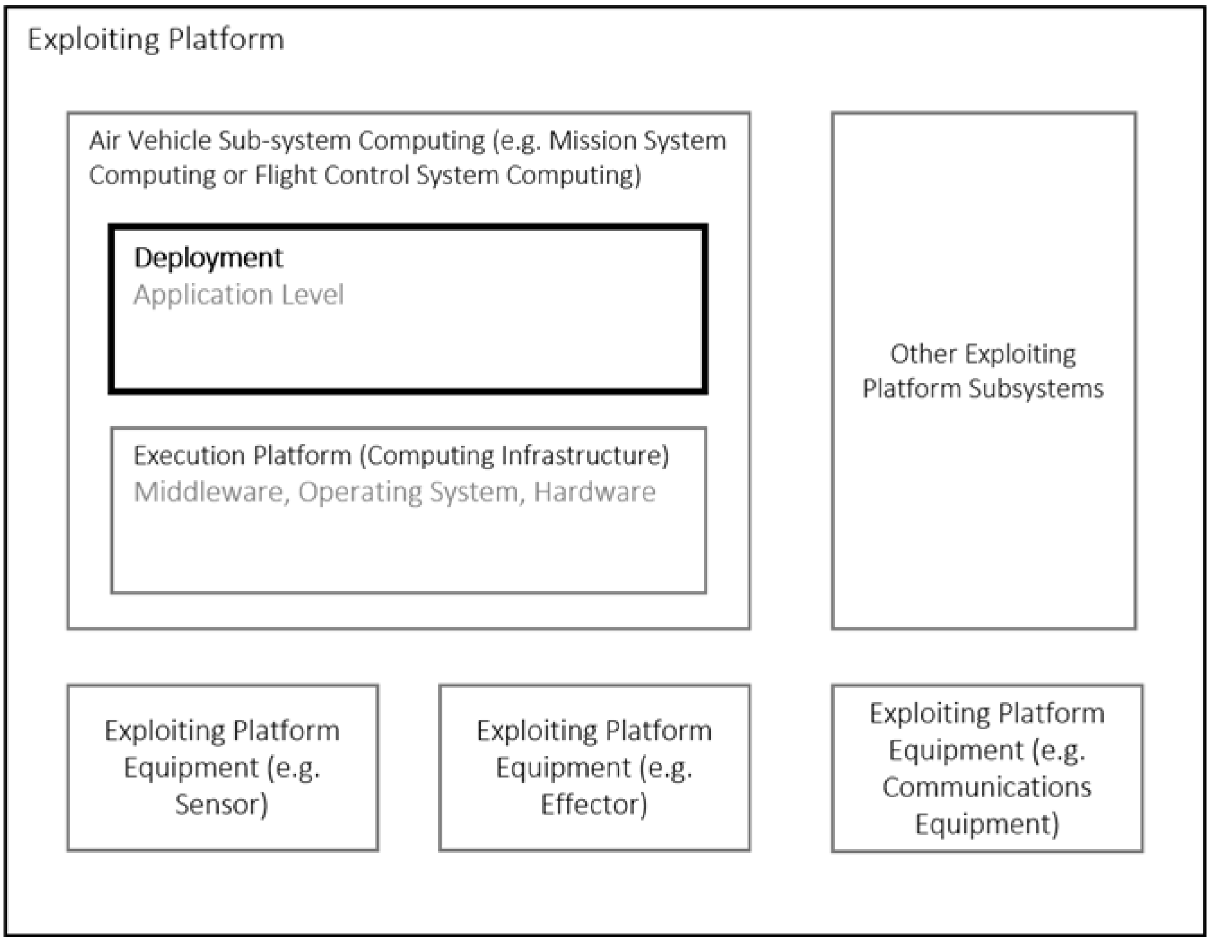

Exploiting Platform
Air Vehicle Sub-system Computing (e.g. Mission System Computing or Flight Control System Computing)
Deployment
Application Level
Execution Platform (Computing Infrastructure)
Middleware, Operating System, Hardware
Other Exploiting Platform Subsystems
Exploiting Platform Equipment (e.g. Sensor)
Exploiting Platform Equipment (e.g. Effector)
Exploiting Platform Equipment (e.g. Communications Equipment)

Figure 214: Example Scope of Deployment within an Exploiting Platform

Note: The lifecycle outlined in this appendix may differ from that used in a specific deployment activity. This appendix does not mandate any particular process and it is down to PYRAMID Exploiters to decide which of the considerations outlined would best suit their own processes.

With future air systems being expected to be increasingly complex, this appendix addresses the issue by taking a “separation of concerns” approach. Instead of breaking down the problem space into a hierarchy of ever decreasing pieces, it breaks the problem space down into two stages:

- The platform independent stage of the design, independent of the Exploiting Platform’s infrastructure, described by the centre section of Figure 215: Deployment Lifecycle Diagram.
    - Platform Independent System Design provides guidance on how the PYRAMID Reference Architecture (PRA) can be used to assist in developing a Platform Independent Model (PIM) and component specifications.
    - Platform Independent Component Design provides guidance on how the PRA can assist in further component maturation by considering component behaviour based on the component specification.
- The platform specific stage of design, usually dependent on the Exploiting Platform’s infrastructure and environment (also known as the execution platform), is described by the lower section of Figure 215: Deployment Lifecycle Diagram.

- Platform Specific System Development provides guidance on how the PRA can assist in the platform design decisions, and in dealing with impacts on the components that arise upon identification of the detailed infrastructure of the Exploiting Platform.
- Platform Specific Component Development provides guidance on how the PRA can assist in further component development for the identified Exploiting Platform's infrastructure.

The appendix further distinguishes between what is required from the provider of a component specification, and the component implementation. This, combined with an iterative design process, allows these concerns to be considered independently even though they may impact on each other in some ways.

The sections in this appendix correspond to a potential deployment lifecycle, where a group of components are taken from the abstract level covered in the PRA through to a final deployed system, as summarised in Figure 215: Deployment Lifecycle Diagram.

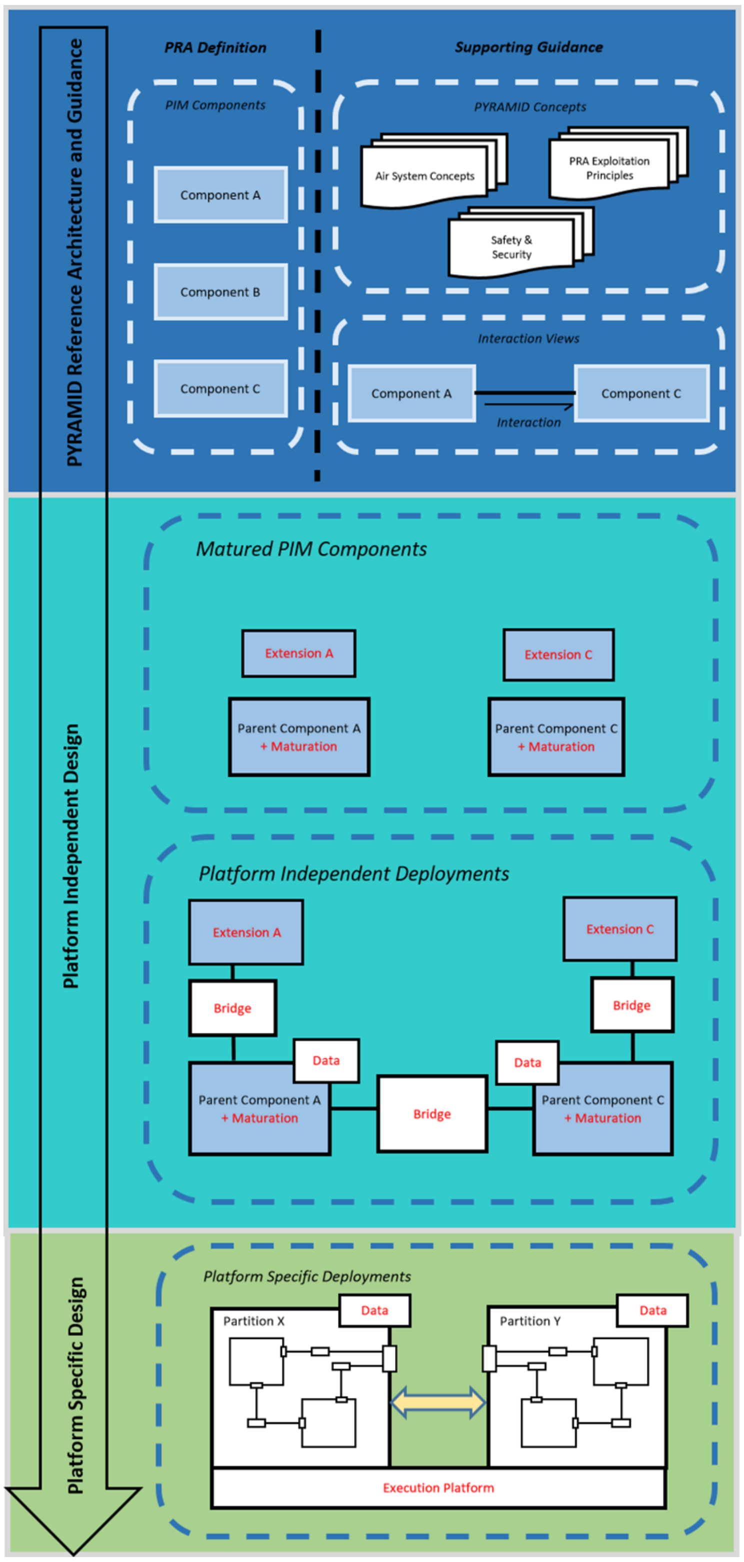


Figure 215: Deployment Lifecycle Diagram

D.1.1 Structure

This appendix has the following sections:

D.1.1.1 Platform Independent System Design

The Platform Independent System Design section provides guidance on the functional analysis of what is required from a deployment, and design of a group of components (or whole system) to provide this functionality, independent of technology or infrastructure detail, with reference to the PRA.

A Platform Independent Model is a representation of a system that is free from implementation (platform) bias, where the term “platform” is used to refer to the infrastructure required to support the execution of application functionality (such as specific operating software and computing hardware), that are irrelevant to the fundamental functionality of the software (Ref. [6]). This computing infrastructure is referred to as the execution platform.

Platform Independent System Design, in the context of the development of a PYRAMID deployment software, is a stage that includes all activities needed to go from a set of requirements on the software deployment to a Platform Independent Model for that deployment.

This involves:

- Identifying which PIM components derived from the PRA to use in the deployment, and developing deployment specific specifications for these components.
- Adding bridges and further deployment specific configuration data.

D.1.1.2 Platform Independent Component Design

The Platform Independent Component Design section provides guidance on the functional analysis and maturation of a component, independent of computing technology or computing infrastructure detail, with reference to the PRA.

Platform Independent Component Design, in the context of the development of a PYRAMID component for a specific deployment, is a stage that includes all activities needed to produce a Platform Independent Model for that component, adding details of internal component behaviour to the component specification produced as part of the Platform Independent System Design.

D.1.1.3 Platform Specific System Development

The Platform Specific System Development section provides guidance on the analysis and development of a system (or group of components) which is suitable for deployment on an Exploiting Platform’s infrastructure, with reference to the PRA.

A Platform Specific Model comprises all the functionality expressed in the equivalent Platform Independent Model with the added design concerns of the realising platform, where the term “platform” is again used to refer to the execution platform - the technological and engineering details that are irrelevant to the fundamental functionality of the software (Ref. [6]).

Platform Specific System Development, in the context of the development of a deployment of PRA-based software, is a stage that includes all activities needed to go from a Platform Independent Model to a Platform Specific Model for that deployment.

This includes:

- Determining the impact on the design of the infrastructure architecture defined for the deployment.
- Ensuring that the Platform Specific Model addresses any requirements for the deployment which need to take account of the execution platform, which may include:
    - Performance requirements that need to take account of processing speeds and communication delays in the system;
    - Requirements related to safety or security;
    - The requirement to be able to load data into the system in order to configure components as necessary, for example by providing the data required for a data-driven component.

D.1.1.4 Platform Specific Component Development

The Platform Specific Component Development section provides guidance on the analysis and development of a component for an Exploiting Platform’s infrastructure, with reference to the PRA.

Platform Specific Component Development, in the context of the development of a PYRAMID component for a specific deployment, is a stage that includes all activities needed to produce a Platform Specific Model for that component.

Component implementation and system integration are not covered at length by this appendix. However, considerations related to component implementation have been grouped together with Platform Specific Component Development.

D.1.1.5 Different Versions of the PRA

The Different Versions of the PRA section, describes some of the potential impacts that may result when updated versions of the PRA are issued, and provides examples and guidance on how this might be managed by Exploiting Programmes.

D.1.2 How to Read the Deployment Guidance

Those involved in development of a PYRAMID deployment should familiarise themselves with the PYRAMID concepts (see Appendix A: PYRAMID Concepts), PRA components (see PYRAMID Technical Standard, Ref. [1], section Component Definitions), and example component build sets in the form of PYRAMID interaction views (IVs) (see Appendix C: Interaction Views). Many of the PYRAMID concept are referred to throughout the Deployment Guide, and these can be read in Appendix A: PYRAMID Concepts.

The following subsections highlight the sections of the guide that are relevant to different engineering roles. The dark blue items are sections of highest relevance, while the light blue items are less relevant but still contain useful guidance. Section, Different Versions of the PRA, is relevant to all engineering roles and is not specifically highlighted in any of the subsections below.

It should be noted that guidance given in the “Good Practice” sections throughout the appendix is included in the section where it is most relevant, but that this does not mean that it is only relevant to that section. Many items of “good practice” will be relevant at multiple stages of the design and development of the deployment.

D.1.2.1 Guidance for System Architects and Designers

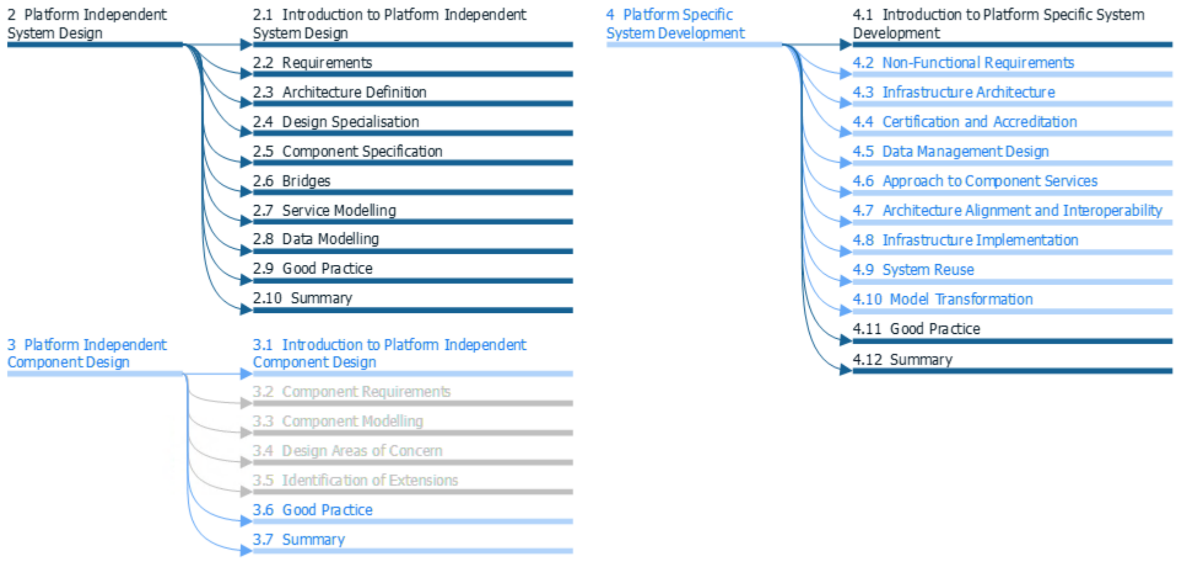

2 Platform Independent System Design
2.1 Introduction to Platform Independent System Design
2.2 Requirements
2.3 Architecture Definition
2.4 Design Specialisation
2.5 Component Specification
2.6 Bridges
2.7 Service Modelling
2.8 Data Modelling
2.9 Good Practice
2.10 Summary
3 Platform Independent Component Design
3.1 Introduction to Platform Independent Component Design
3.2 Component Requirements
3.3 Component Modelling
3.4 Design Areas of Concern
3.5 Identification of Extensions
3.6 Good Practice
3.7 Summary
4 Platform Specific System Development
4.1 Introduction to Platform Specific System Development
4.2 Non-Functional Requirements
4.3 Infrastructure Architecture
4.4 Certification and Accreditation
4.5 Data Management Design
4.6 Approach to Component Services
4.7 Architecture Alignment and Interoperability
4.8 Infrastructure Implementation
4.9 System Reuse
4.10 Model Transformation
4.11 Good Practice
4.12 Summary

Figure 216: Main Sections of Interest for System Architects and Designers

System Architects and Designers are responsible for creating the design and architecture of a deployment by describing its structure, behaviours and interactions. The starting point for those with the role of designing a deployment is the Platform Independent System Design section. Platform Independent System Design considers the overarching functional considerations for the deployment and drives out the specifications for the individual components. Some of the technology dependent approaches covered in the Platform Specific System Development section will also be of interest to System Architects and Designers.

D.1.2.2 Guidance for Component Designers

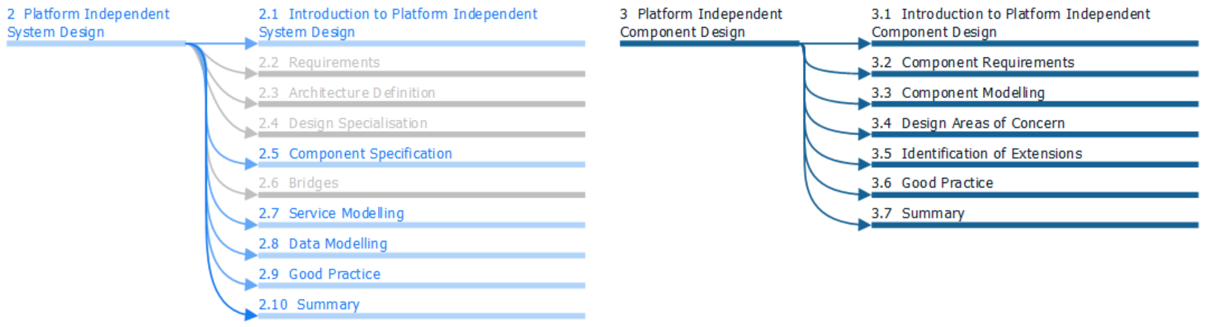

2 Platform Independent System Design
2.1 Introduction to Platform Independent System Design
2.2 Requirements
2.3 Architecture Definition
2.4 Design Specialisation
2.5 Component Specification
2.6 Bridges
2.7 Service Modelling
2.8 Data Modelling
2.9 Good Practice
2.10 Summary
3 Platform Independent Component Design
3.1 Introduction to Platform Independent Component Design
3.2 Component Requirements
3.3 Component Modelling
3.4 Design Areas of Concern
3.5 Identification of Extensions
3.6 Good Practice
3.7 Summary

Figure 217: Main Sections of Interest for Component Designers

A Component Designer is responsible for the design of a component based on a component specification, which may have been produced as part of an overall system design for a deployment. The starting point for a designer of a component using the PRA is the component specification design (see Component Specification section), so it will be useful for Component Designers to be aware of the work done to produce this specification.

The Platform Independent Component Design section describes the bulk of the component design work, which can be progressed once the component requirements and services have been specified.

This section outlines the overarching functional considerations for components and defines the work that is expected to be done during component design.

D.1.2.3 Guidance for Component Developers

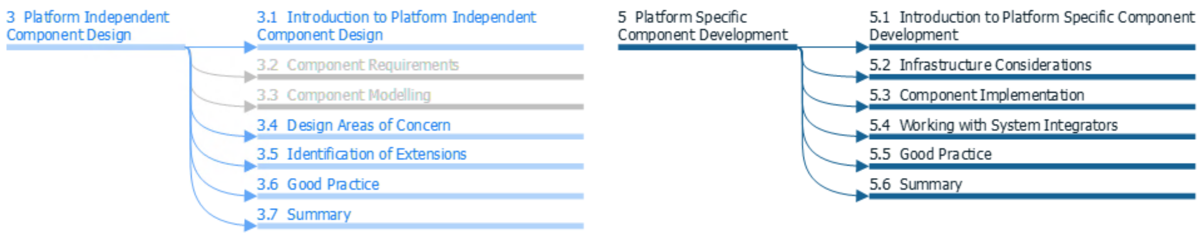

3 Platform Independent Component Design
3.1 Introduction to Platform Independent Component Design
3.2 Component Requirements
3.3 Component Modelling
3.4 Design Areas of Concern
3.5 Identification of Extensions
3.6 Good Practice
3.7 Summary
5 Platform Specific Component Development
5.1 Introduction to Platform Specific Component Development
5.2 Infrastructure Considerations
5.3 Component Implementation
5.4 Working with System Integrators
5.5 Good Practice
5.6 Summary

Figure 218: Main Sections of Interest for Component Developers

A Component Developer is responsible for the development of a component tailored for a specific execution platform. The starting point for a developer of a component using the PRA is the Platform Specific Component Development section. This section outlines the steps to turn the previously developed Platform Independent Component Design into components that will work on the target execution platform.

D.1.2.4 Guidance for System Integrators

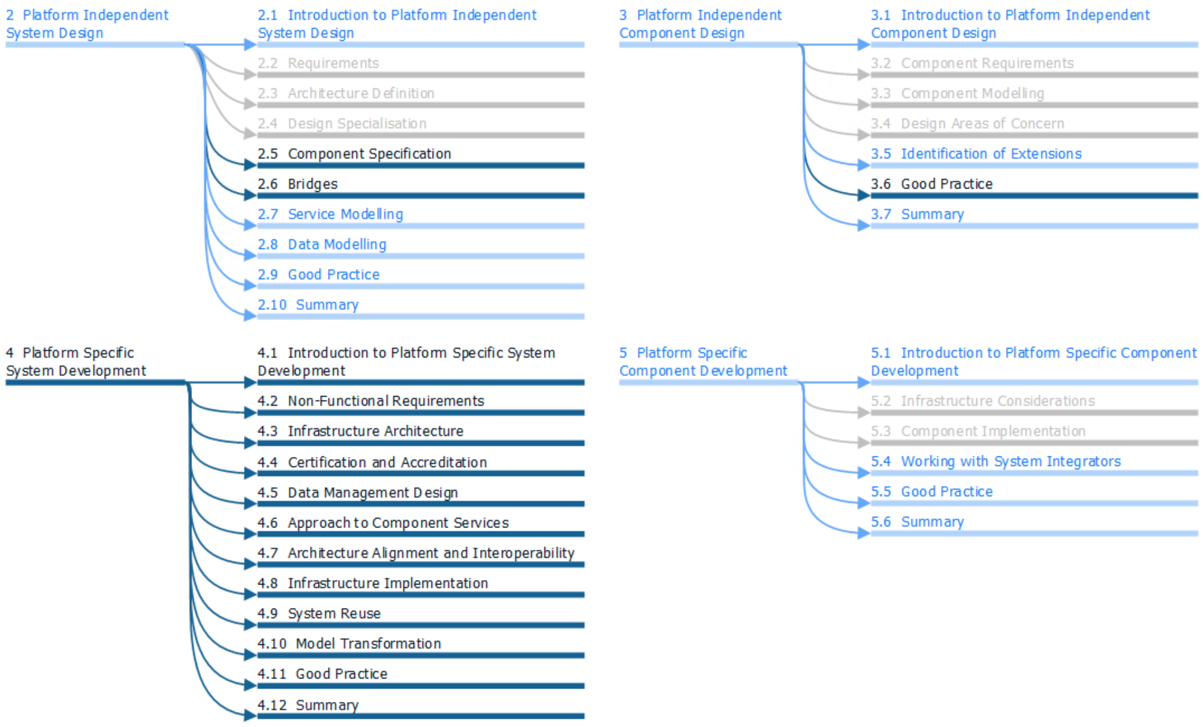

2 Platform Independent System Design
2.1 Introduction to Platform Independent System Design
2.2 Requirements
2.3 Architecture Definition
2.4 Design Specialisation
2.5 Component Specification
2.6 Bridges
2.7 Service Modelling
2.8 Data Modelling
2.9 Good Practice
2.10 Summary
3 Platform Independent Component Design
3.1 Introduction to Platform Independent Component Design
3.2 Component Requirements
3.3 Component Modelling
3.4 Design Areas of Concern
3.5 Identification of Extensions
3.6 Good Practice
3.7 Summary
4 Platform Specific System Development
4.1 Introduction to Platform Specific System Development
4.2 Non-Functional Requirements
4.3 Infrastructure Architecture
4.4 Certification and Accreditation
4.5 Data Management Design
4.6 Approach to Component Services
4.7 Architecture Alignment and Interoperability
4.8 Infrastructure Implementation
4.9 System Reuse
4.10 Model Transformation
4.11 Good Practice
4.12 Summary
5 Platform Specific Component Development
5.1 Introduction to Platform Specific Component Development
5.2 Infrastructure Considerations
5.3 Component Implementation
5.4 Working with System Integrators
5.5 Good Practice
5.6 Summary

Figure 219: Main Sections of Interest for System Integrators

System Integrators are responsible for integrating the components to create a deployment. Those whose role includes integrating the deployment are likely to be interested to some extent in all the work that is to be undertaken. This role may be covered by a party acting as both designer and integrator, although it may be undertaken as a distinct role separate to the System Architect. Earlier integrator involvement will lead to a smoother integration and acceptance, particularly if the integration authority is different to the system design authority, and this should especially be considered for work described in the Component Specification, Bridges and Certification and Accreditation sections.

D.1.3 Overall Deployment Approach

In general, the PYRAMID approach described in this appendix is not a simple linear or 'waterfall' process, where one stage can be completed before moving on to another. The underlying principle of 'separation of concerns' (described in the main body of the PYRAMID Technical Standard Guidance document, Separation of Concerns Defined by the PRA section) means that the platform independent and platform specific parts of system development should be considered separately, but not in isolation from each other. There are information flows between development stages and teams not only in the directions that might be expected, i.e. from system design to component design and from platform independent design to platform specific development, but also in the opposite directions. Some, but not all, of these potential information flows are shown in Figure 220: Example information flows between development stages.

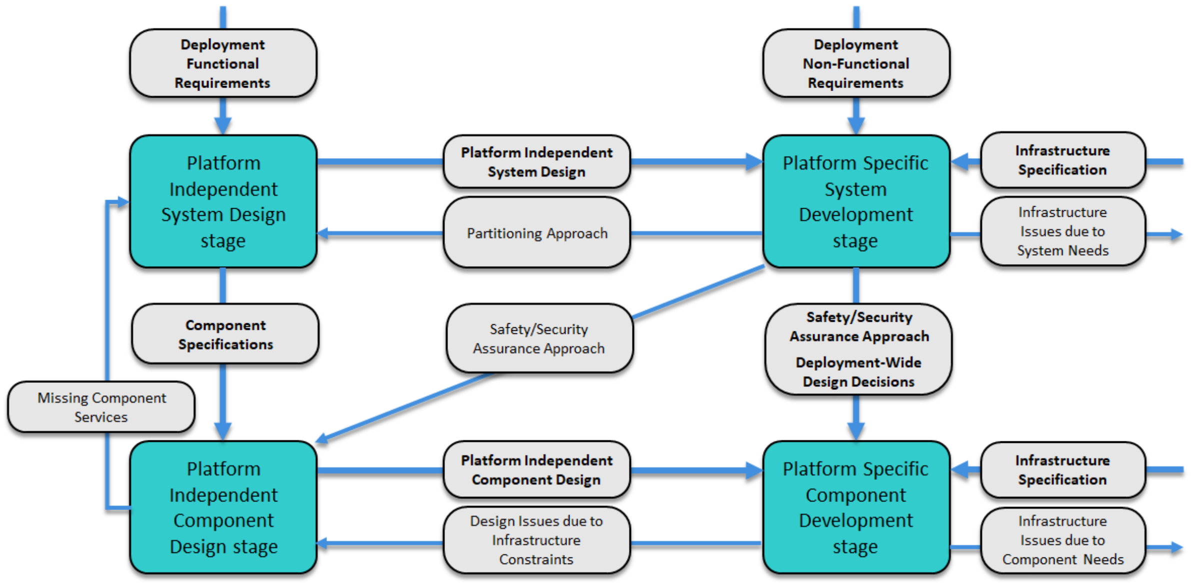

Deployment Functional Requirements
Deployment Non-Functional Requirements
Platform Independent System Design stage
Platform Independent System Design
Partitioning Approach
Platform Specific System Development stage
Infrastructure Specification
Infrastructure Issues due to System Needs
Component Specifications
Safety/Security Assurance Approach
Safety/Security Assurance Approach
Deployment-Wide Design Decisions
Missing Component Services
Platform Independent Component Design stage
Platform Independent Component Design
Design Issues due to Infrastructure Constraints
Platform Specific Component Development stage
Infrastructure Specification
Infrastructure Issues due to Component Needs

Figure 220: Example information flows between development stages

New items of information from subsequent or parallel development stages may mean that a new iteration of an output from a design or development stage is required.

D.2 Platform Independent System Design

D.2.1 Introduction to Platform Independent System Design

This section covers the Platform Independent System Design. 'Platform Independent' in this context refers to consideration of what functionality is required, with independence from the execution platform (the infrastructure and computing hardware supporting the application functionality). This stage is summarised in Figure 221: Platform Independent System Design Stage.

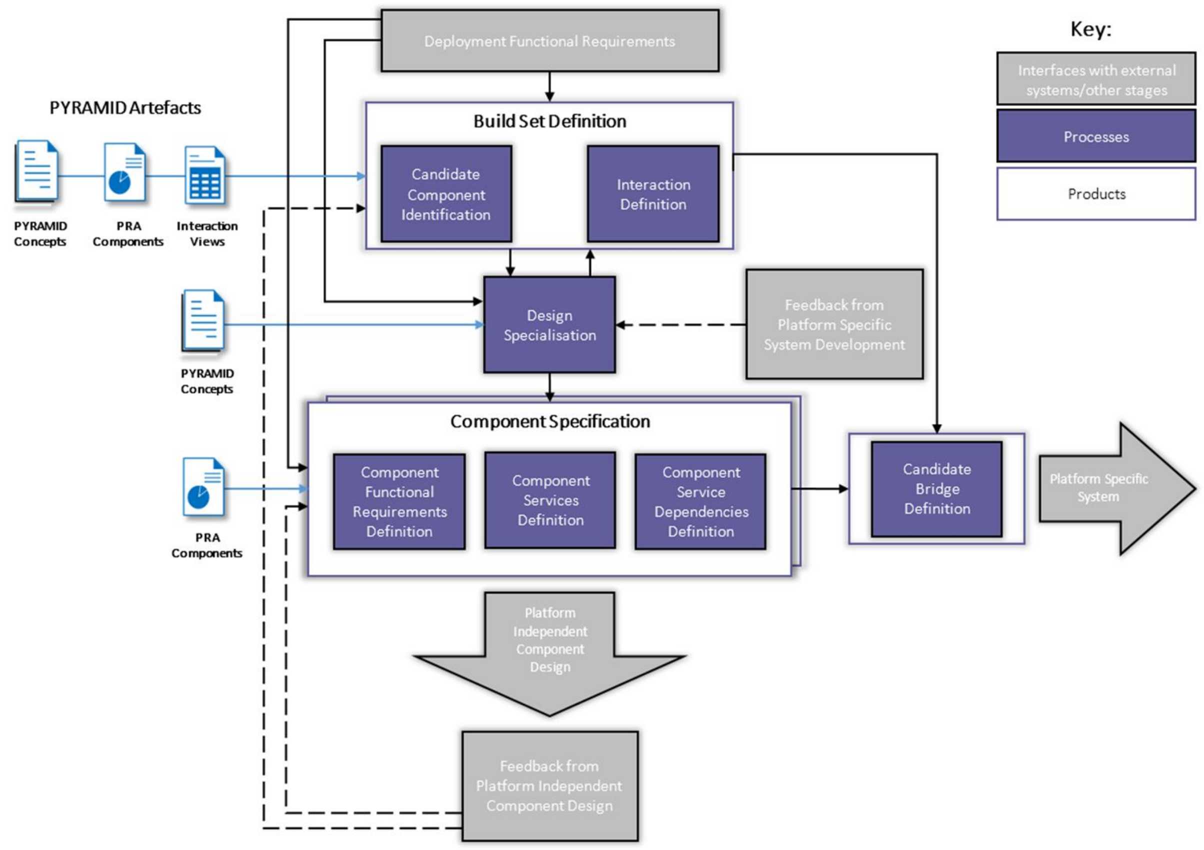

Key:
Interfaces with external systems/other stages
Processes
Products
Deployment Functional Requirements
PYRAMID Artefacts
PYRAMID Concepts
PRA Components
Interaction Views
Build Set Definition
Candidate Component Identification
Interaction Definition
Design Specialisation
Feedback from Platform Specific System Development
PYRAMID Concepts
Component Specification
Component Functional Requirements Definition
Component Services Definition
Component Service Dependencies Definition
Candidate Bridge Definition
Platform Specific System
PRA Components
Platform Independent Component Design
Feedback from Platform Independent Component Design

Figure 221: Platform Independent System Design Stage

D.2.2 Requirements

In order to produce a Platform Independent System Design for a specific PYRAMID deployment, the requirements for this deployment must first be known. Although covered in this section, determination of deployment requirements needs to take place before Platform Independent System Design can begin.

Figure 214: Example Scope of Deployment within an Exploiting Platform shows the scope of a deployment within the wider context of the Exploiting Platform. In order to derive a set of requirements for the PYRAMID deployment from higher-level requirements sets applicable to the wider Exploiting Platform, normal systems engineering processes should be used. During these processes, non-functional requirements from higher levels can result in functional requirements which apply to the PYRAMID deployment. For example, a non-functional requirement for a certain level of availability from the mission system may result in a functional requirement for the deployment software to monitor and record system availability. Figure 222: Example Derivation of Deployment Requirements from Exploiting Platform Requirements illustrates how deployment requirements may be derived from higher-level requirement sets, using an example of a sub-system that contains the parts of the system developed using PYRAMID components.

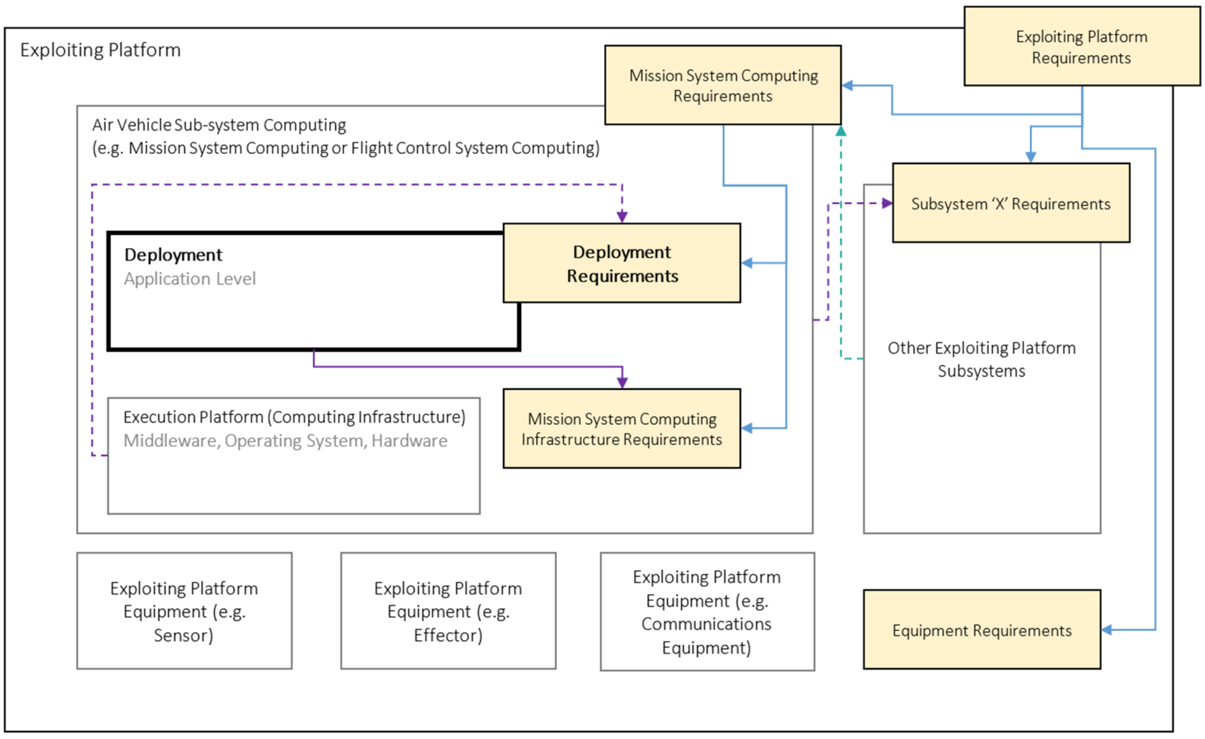

Exploiting Platform
Exploiting Platform Requirements
Mission System Computing Requirements
Air Vehicle Sub-system Computing
(e.g. Mission System Computing or Flight Control System Computing)
Subsystem 'X' Requirements
Deployment
Application Level
Deployment Requirements
Other Exploiting Platform Subsystems
Execution Platform (Computing Infrastructure)
Middleware, Operating System, Hardware
Mission System Computing Infrastructure Requirements
Exploiting Platform Equipment (e.g. Sensor)
Exploiting Platform Equipment (e.g. Effector)
Exploiting Platform Equipment (e.g. Communications Equipment)
Equipment Requirements

Figure 222: Example Derivation of Deployment Requirements from Exploiting Platform Requirements

Generally, the ‘first pass’ through Platform Independent System Design will only need to consider the functional requirements that apply to the PYRAMID deployment, which will include any functionality needed from the application level software, including functionality that allows the fulfilment of non-functional requirements from higher levels (for example a requirement to monitor and record system availability).

Non-functional requirements at the deployment level, which do not specify any behaviour or functionality that will be required from the deployment but are only concerned with how the software needs to run on the computing infrastructure, will be initially considered during Platform Specific development stages. Subsequent updates to the Platform Independent System Design will need to consider input from Platform Specific development stages, that may include input based on non-functional requirements. For example, safety and security requirements considered in the Platform Specific System Design may result in a need to create multiple variants or instances of a component within the build set in the Platform Independent System Design (see the Multiple Instances of Components and Certification and Accreditation sections).

Note that it is expected that the specific elements of a deployment that are intended to comply with the PRA, and the version of the PRA that those elements are intended to comply with, would be agreed as part of the requirements for the deployment (see section, Different Versions of the PRA).

D.2.3 Architecture Definition

The PRA does not provide an Exploiter with a system architecture. The first step in producing a Platform Independent System Design is to use the information provided in the PRA, and accompanying guidance, to produce an architecture that meets the deployment’s requirements by identifying the PRA components, and the interactions between them, that will be needed.

One approach that may be helpful is to produce a schematic overlaying features, functionality and known hazards across the deployment components. Considering all of these factors at this stage may help reduce the risk of costly system redesign later in the design lifecycle.

D.2.3.1 Build Set Identification

The first stage of defining an architecture for a deployment is to identify the PRA components that will be needed. These can be captured within a PIM build set - effectively a parts list that specifies the selected components that make up a specific PYRAMID deployment and their interconnections.

An example diagram for a build set of components is shown in Figure 223: Example Build Set Diagram. Multiple tiers of build set may be required to capture a complete deployment.

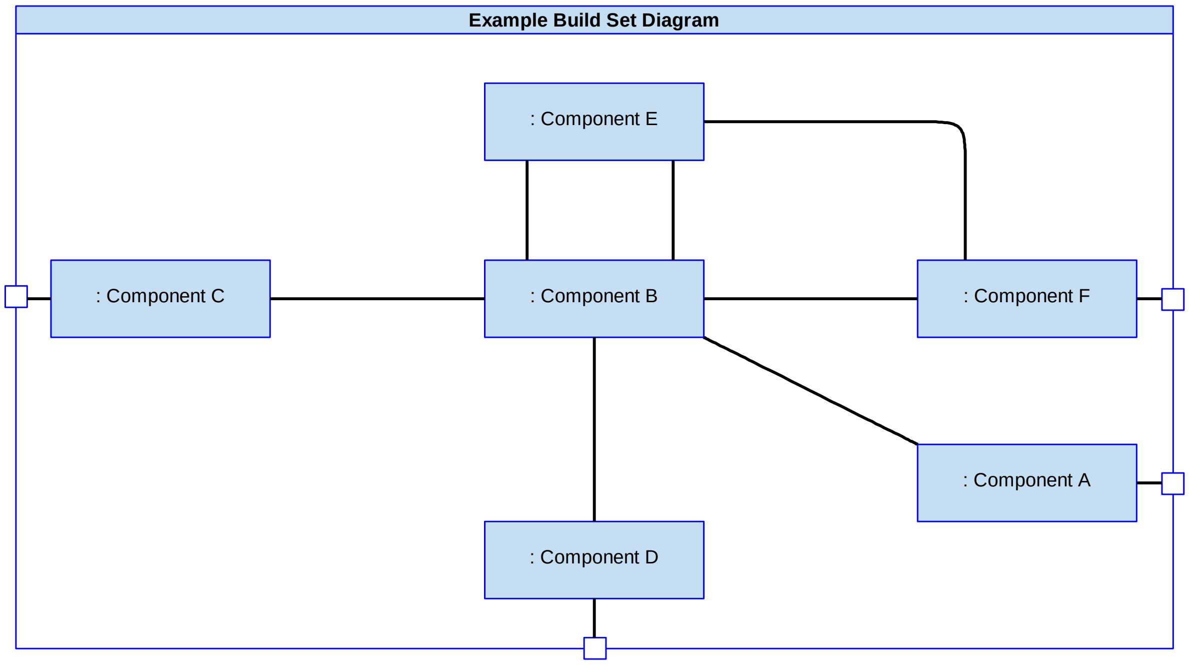


Figure 223: Example Build Set Diagram

In order to identify the PRA components that can be used to satisfy the functional requirements and should therefore be included in the build set, a mapping can be made between the requirements and the component responsibilities associated with each component. However, it is important not to read these responsibilities in isolation. The following artefacts within the PYRAMID Technical Standard, Ref. [1] and PYRAMID Technical Standard Guidance document can be used to provide context regarding the component responsibilities and ensure Exploiters interpret them correctly:

- The Role and Overview for each component, which state the component's overall purpose and include examples of use. These are contained within the PYRAMID Technical Standard, Ref. [1], section Component Definitions.
- Semantics Diagrams within each PRA component definition, which contain entity relationships that identify the subject matter and scope of the component. These are contained within the PYRAMID Technical Standard, Ref. [1], section Component Definitions.

- Service Definitions, which define a basic set of services for each PRA component. These definitions can assist in determining how build set components will interact. These are contained within the PYRAMID Technical Standard, Ref. [1], section Component Definitions.
- The Component Composition, which contains information about responsibilities and services, which may apply to any component but have not been included within individual component definitions. The Component Composition is included in the PYRAMID Technical Standard, Ref. [1].
- Air System PYRAMID Concepts, many of which discuss architecture-wide approaches to the combination of components to provide capability in specific functional areas. These PYRAMID concepts are included in Appendix A: PYRAMID Concepts, section Air System PYRAMID Concepts.
- Interaction views (IVs), which show examples of how higher level functions may be achieved using a combination of components. The IVs are included in Appendix C: Interaction Views.

The initial build set defined is likely to represent a logical architecture, showing which PRA components will be needed to provide the required functionality. Following consideration of component multiplicity, which takes place as part of design specialisation (see the Design Specialisation section); the build set may need to be updated to represent the actual variants and instances of components which will be part of the deployment.

D.2.3.2 Interaction Definition

As part of the platform independent design stage, the interactions between different components in a PIM build set need to be elaborated. Multiple techniques for identifying the initial deployment-specific interactions can be used, for example use case analysis based on use cases that represent the functionality required from the deployment. Figure 224: Example Communications Diagram showing Component Interactions shows a communication diagram that shows interactions for the example introduced in Figure 223: Example Build Set Diagram, as an example of how component interactions could be represented.

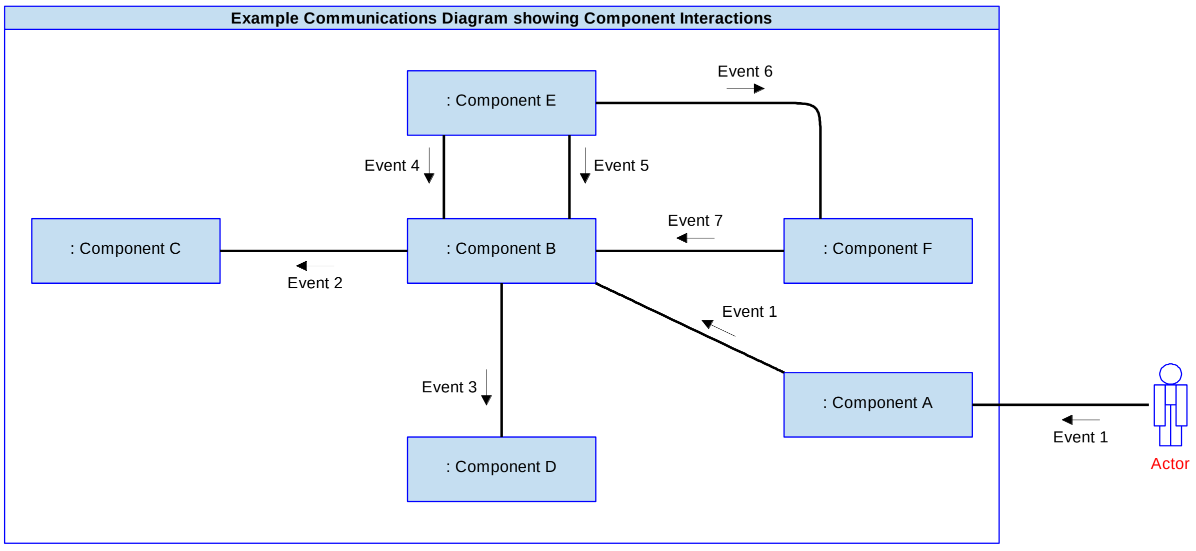


Figure 224: Example Communications Diagram showing Component Interactions

D.2.4 Design Specialisation

Once a component has been selected and its interactions with other components defined, the next step is to consider how it should be specialised and developed to produce a component specification.

Component design specialisation is the process of considering how the generic component definition provided as part of the PRA should be adapted in order to meet the specific needs of the deployment. Not every implemented PYRAMID component will need to cover the whole scope of subject matter or responsibilities included in the PRA component definition they are based on. The process of component specialisation should allow Exploiters to understand what specific parts of the component scope need to be included in the PYRAMID component(s) that they are developing. Further development will then be needed in order to produce a component specification containing sufficient detail to allow Platform Independent Component Design.

Component design specialisation should be implemented in accordance with the PYRAMID concepts, such as the Component Extensions and Data Driving PYRAMID concepts (see Appendix A: PYRAMID Concepts, sections Component Extensions and Data Driving).

The following subsections detail some aspects on how components can be specialised to meet the specific requirements of a deployment, but are not intended to be exhaustive. Additional factors may need to be taken into consideration, depending upon the overall system design and development methodology being followed.

D.2.4.1 Autonomy and System Constraints

The level of autonomy that is desired in the system, and the handling of system constraints, should be considered as part of the Platform Independent System Design and will affect the functionality that is required from components. The following PYRAMID concepts may be of use in determining the component functionality that is required:

- The **Control Architecture** PYRAMID concept (Appendix A: PYRAMID Concepts, section Control Architecture) describes a control architecture that has been embodied in the PRA, enabling the architect of an Exploiting Platform to implement system-wide control.
- The **Autonomy** PYRAMID concept (Appendix A: PYRAMID Concepts, section Autonomy) provides recommendations on capturing the level of autonomy that is deemed appropriate.
- The **Constraint Management** PYRAMID concept (Appendix A: PYRAMID Concepts, section Constraint Management) provides recommendations on handling constraints associated with rules and limits, etc.
- The **Capability Management** PYRAMID concept (Appendix A: PYRAMID Concepts, section Capability Management) describes how changes in component and wider system capability should be detected and managed, and the use of capability information in determining responses to the changes.

D.2.4.2 Component Scaling

For a given deployment, it may be desirable to specialise the functionality of components for resourcing, temporal or spatial reasons (e.g. offline planning or ground support). Each component can be scaled into distinct component variants, which are specialised to cover a particular subset of the component's subject matter or responsibilities, or to provide the specific functionality or services required for the circumstances in which the component will be used, such as the relevant operational context or resource profile.

When scaling a component it is important to avoid producing high coupling between the component variants. Given that variants will share the subject matter knowledge of their original component, this will require some thought. Meaningful and comprehensive boundaries between the variants are needed in order to ensure the components can operate without each other, or so that they can co-operate efficiently.

Component variants may be used to distribute a component's functionality to address non-functional requirements including resourcing, security or safety concerns. If this is the reason for the variants, they are likely to need synchronising with each other (see the Data Synchronisation section). Additional guidance on distributing components by context is provided by M. Page-Jones who proposes several key strategies for developing coherent components. See Refs. [18], [33], [34] and [35].

D.2.4.3 Multiple Instances of Components

Whereas scaling a component can enable specialisation of its overall functionality for aspects like different operational situations or settings (see the Component Scaling section), there may be a need to use multiple instantiations of the same implemented PYRAMID component. This may be done for numerous reasons, for example to:

- Distribute behaviour for resourcing, security and/or safety reasons, in order to address non-functional requirements covering these areas. This will allow compartmentalisation of information for security reasons, and allow the introduction of redundancy for safety. See the Certification and Accreditation section.
- Instantiate a data-driven generic component multiple times for specific purposes.

The communications between multiple instances of components need to be kept to a minimum. The services provided by a component to allow interactions between multiple instances of itself must be as carefully considered as those between different component variants.

The following PYRAMID concepts offer recommendations on distributed working, which may require multiple instances of the same component:

- The **Multi-Vehicle Coordination** PYRAMID concept (Appendix A: PYRAMID Concepts, section Multi-Vehicle Coordination) contains guidance on the orchestration of tasking over a system distributed between multiple geographic locations;
- The **Use of Communications** PYRAMID concept (Appendix A: PYRAMID Concepts, section Use of Communications) contains guidance on how the data communications work together to provide a managed multi-node solution;
- The **Operational Support** PYRAMID concept (Appendix A: PYRAMID Concepts, section Operational Support) contains guidance on how instances of components can be used in the planning and operational usage of the system.

Where instances of the same component co-exist, they are most likely to be changed by *configuration* in order to meet the particular requirements specific to the role they play within the Exploiting

Platform. An example might be multiple instances of a managed resource component with each instance configured to operate as a specific type of resource (e.g. a sensor type). For more details see the Data Driving PYRAMID concept (Appendix A: PYRAMID Concepts, section Data Driving).

Note that using multiple instantiations of a generic component for a particular purpose is a choice. A single parent instance could instead be used as the final arbiter of its extension components. For example, a single instance of a managed resource component could delegate behaviour for different resource types to the extension components responsible for behaviour specific to that resource type. For information on component extensibility, see the Identification of Extensions section and the Component Extensions PYRAMID concept (Appendix A: PYRAMID Concepts, section Component Extensions).

D.2.5 Component Specification

Component specifications are needed in order to define what the purpose of each component is, what it provides to the overall deployment and what it needs from the deployment. This information is required for the System Independent Component Design stage of the deployment to proceed, in which the component designs can be developed in parallel.

Component specifications will be based on:

- The build set for the deployment, showing the interactions that a component has with other components;
- The functional requirements for the deployment;
- Analysis that has taken place as part of the design specialisation stage, to determine how the generic PRA definition of a component should be specialised to cover the specific subject matter and provide the specific functionality required for a deployment;
- Feedback from the Platform Independent Component Design stage.

As feedback from the Platform Independent Component Design stage will only be available once an initial specification has been produced to enable the Platform Independent Component Design to begin, this means that multiple iterations may be required to refine component specifications.

In addition to how the component will contribute to the required deployment functionality, the component specification should also specify what will be required from the component in order that:

- The PYRAMID component will comply with the relevant PRA component definition, see PYRAMID Technical Standard, Ref. [1], section Component Definitions.
- The PYRAMID component can contribute to the deployment as a whole, informed by PYRAMID concepts, see Appendix A: PYRAMID Concepts.

D.2.5.1 Component Specifications and Reuse

Depending on the level of reuse being undertaken in the deployment, a component specification may already exist. Reuse of components should be maximised at this stage so that software and artefacts with pre-existing functionality are not developed unnecessarily.

D.2.5.2 Contents of Component Specifications

The exact contents of a component specification may depend on any wider system development methodology that is being used by the Exploiting Programme (see the Service Modelling and Data Modelling sections), but they will typically cover three main areas:

- **Functional requirements** - see the Functional Requirements section;
- **Services**, which define what is provided by the component to the overall deployment and what it requires in return - see the Services section;
- **Service dependencies**, which define the relationships between what the component provides and what it requires from the rest of the deployment - see the Service Dependencies section.

D.2.5.2.1 Functional Requirements

Component requirements need not be textual. One of the key goals of model based design is to provide a more automated way of progressing from high level requirements to running software. Ideally, to support this the requirements should be captured in a machine interpretable way, with the component supplier and System Integrator having an agreed approach to electronic exchange of requirements, traceability and verification evidence.

D.2.5.2.2 Services

Services are at the core of the component specification. A service may be as simple as providing a piece of information or as complex as generating a complete flight path. There are many approaches to defining the services within components - while the PRA does not mandate any particular methodology for service design the PRA has been designed with support for the following modelling methodologies in mind:

- A service modelling approach as detailed in the Service Modelling section.
- A data modelling approach as detailed in the Data Modelling section.
- A domain modelling approach as detailed in Ref. [9].
- Multiple approaches can be used at once, but often a dominant method is required across a deployment to avoid multiple definitions of the same service; for more on this see Ref. [5]. Each PRA component definition (included within the PYRAMID Technical Standard, Ref. [1], section Component Set) provides basic services which can be used as the basis for the services in a component specification.

D.2.5.2.3 Service Dependencies

A service dependency explains how the services a component provides depend on other provided services and consumed services. The dependencies between a component’s service activities can be represented in activity diagrams, of which a generalised version is shown in Figure 225: Service Dependency Diagram. Aspects such as events and signals can be included in an activity diagram to provide the desired level of detail.

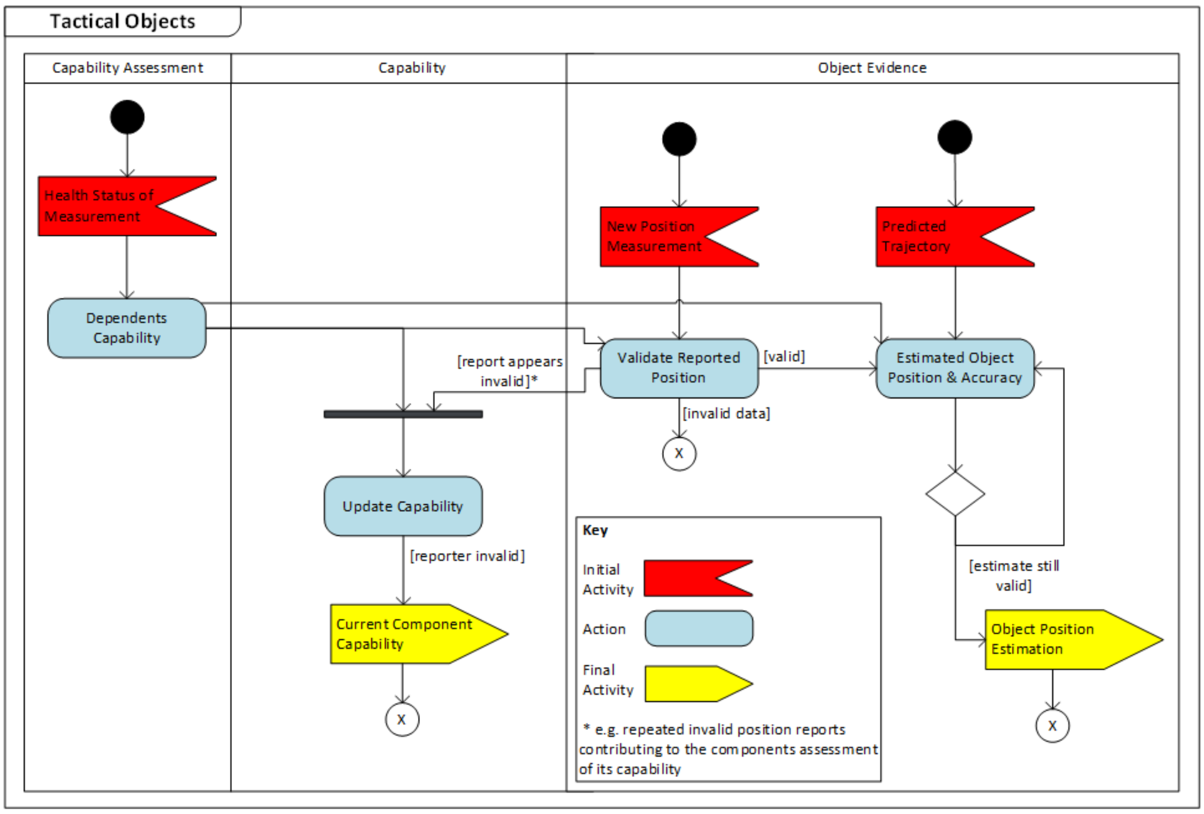

Tactical Objects
Capability Assessment
Capability
Object Evidence
Health Status of Measurement
Dependents Capability
[report appears invalid]*
Update Capability
[reporter invalid]
Current Component Capability
X
New Position Measurement
Predicted Trajectory
Validate Reported Position
[valid]
[invalid data]
X
Estimated Object Position & Accuracy
[estimate still valid]
Object Position Estimation
X
Key
Initial Activity
Action
Final Activity
* e.g. repeated invalid position reports contributing to the components assessment of its capability

Figure 225: Service Dependency Diagram

As the components are refined during a deployment, these diagrams should be modified and matured. The PRA includes a service dependency activity diagram in each component definition, and these can be used to inform a component's service dependencies for a deployment.

A service dependency is the part of the component specification that has to be the most co-operative in its development, as it is used to tell a System Integrator what has to be provided to the component in order to achieve the service that is requested.

D.2.6 Bridges

Information needs to be exchanged between the components in a deployment. Direct interfaces between components are generally not possible, as:

- Components within a PYRAMID deployment are deliberately scoped to be completely independent of each other, which has many benefits including promoting component reuse (see Appendix A: PYRAMID Concepts, section Component Connections for more details), so do not share interface definitions;
- There is a "semantic gap" between components - their understanding of the subject of an information exchange is based on their own subject matter, and therefore information transformations may be required between components in order to 'translate'.

Information is therefore passed using inter-component connections known as bridges.

Part of the design approach is to identify these bridges to make sure that the semantic gap being crossed is not too wide, and that functionality that needs to be covered by the deployment is not being hidden in the bridges. If a particular design does require a bridge to carry out complex reasoning, this can be a sign that an additional component, covering the subject matter that is being reasoned about, should be added to the build set.

Within the Platform Independent Model produced by the Platform Independent System Design, the impact of the execution platform on timing and the partitioning of the components across different infrastructure elements is not considered. The model ‘assumes’ that all interactions are instantaneous and allowed. The platform specific modelling described in later sections will address non-functional constraints such as partitioning and timing. If updates to the required logical behaviour are discovered as a result of platform specific analysis, an additional iteration of the Platform Independent System Design may be needed to incorporate these updates.

For more information on bridges, along with some examples of bridges within the standardised connection patterns see Appendix A: PYRAMID Concepts section Component Connections.

D.2.7 Service Modelling

The required activities for Platform Independent System Design may take place within a wider systems development methodology that is being applied by the Exploiting Programme.

Service modelling methodologies, such as Service Oriented Architecture (SOA), assume that the components can be fully specified by their services alone. Developing a Platform Independent System Design using these methodologies should be well supported by the PRA, as the services provided as part of the definition of PRA components will have similar characteristics to the services that need to be defined to produce a component specification using an SOA methodology. For more information on SOA see Ref. [7] and [8].

D.2.7.1 Component Specification Artefacts

Service modelling requires two types of services to be defined at the PIM level:

- **Provided services**, which are services offered by the component;
- **Consumed services**, which are services required by the component from elsewhere in the system.

The following artefacts could be used to define these services:

- **Service interfaces**, which define the functionality provided or consumed by the component. Interfaces may contain service operations and/or attributes that are each related to the overall provided or consumed service.
- **Component centric service contracts** that define what the component is relying on being provided to it, in order that it can provide services.

Service interfaces may be:

- **Simple service interfaces**, where the service interface can be defined by attributes on a class which can either be provided to the component in question by the other participant in the interaction (for a consumed service) or provided by the component to the other participant (for a provided service). The attributes on the class contain the data which is passed between the components.

- **Contract service interfaces**, where the service interface cannot be represented as a simple service interface, for example if the service interface is requesting an activity rather than just passing data. One way of representing a contract service interface would be to define a sequence of interactions between the component and the other participant in the service.

Service interfaces can be combined to create more complex service interfaces - for example a contract service interface may include the exchange of data using a simple service interface.

Service patterns (referred to as component interaction patterns in the Component Connections PYRAMID concept) define common solutions to the need to combine services to provide commonly needed types of capability - for example capabilities which allow the coordination of activities across multiple components. See Appendix A: PYRAMID Concepts, section Component Connections for more details on service patterns and examples of component interaction patterns that may be relevant to PYRAMID deployments.

D.2.7.2 Relationship to PRA artefacts

The PRA component definitions contain definitions for a basic set of services, with simple interfaces, for interacting with other PYRAMID components. Additional service definitions which may apply to any component can be found in the Component Composition. (Not all interactions are covered by the services defined in the PRA, for example interactions with external systems which cross the PRA scope boundary are not covered. For more details see PYRAMID Technical Standard, Ref. [1], section Component Services Not Defined by the PRA.)

These definitions may be used as the basis for the services that form part of the component specification, but it is likely that these will need to be both specialised (in order to align with the specific requirements of the deployment) and refined (in order to add more detail to the basic definitions provided) in order to provide a sufficiently detailed specification on which to base the design of a component.

D.2.8 Data Modelling

Another methodology that may be adopted by the Exploiting Programme is data modelling. Data modelling is focussed on defining a component through its entities, their attributes, and the relationship between the entities. PRA component definitions contain semantics diagrams, which are used to express the scope of the component by defining the entities within it. These are not exhaustive definitions of all entities that may be required by a component, but should provide useful information that could be useful in the development of a data model.

Data modelling can be used as a stand-alone approach or in combination with other approaches, such as alongside the definition of services in order to allow easier integration of components.

D.2.8.1 Model Artefacts

Three particular types of data model may be of interest at the PIM level:

- A **shared data model** - which contains low level data type definitions for entities such as physical measurements. Using a shared data model for these types of data across the deployment can help reduce the amount of data conversion that is needed in bridges.

- A **system information data model** - used to capture relationships between components. See the System Information Data Model section for more information.
- A **component information data model** - used to define a component specification, alongside user concept views and data views. See the Component Information Data Model and Data Views section for more information.

D.2.8.1.1 System Information Data Model

A typical Exploiting Programme might provide the following system-level artefacts:

- **User concept views**, incorporating customer focused system use cases.
- A **system information data model**, describing the subject matter data groups. The subject matter entities within the PRA can be used as a starting point for the data model.

The information model can be used to:

- Model parts of the system, particularly the data-driven parts of a component.
- Define the semantic data exchanges across multiple systems operating in a common operating domain.
- Support reuse of existing data types across multiple deployments prior to contract placement.

In the same way that a component information data model is used in combination with data views to define the component's relationship to its interfaces (see the Component Information Data Model and Data Views section), the system data model can be used to capture the relationship between different components. Again a layered approach here can bring reuse advantages, including defining bridge conversion routines.

D.2.8.1.2 Component Information Data Model and Data Views

A component specification can provide the following artefacts:

- A **component information data model**, defining the data that is used within a component, and **data views,** defining which parts of the entity data model are exchanged over a particular interface.
- **User concept views**, incorporating customer focused component use cases, therefore only showing the parts of the deployment that the component is directly involved in.

An information model is represented as an entity, relationship and attribute diagram, which shows a logical breakdown of the data structure within the component.

The information model can be used to define the component specification, as the understanding of the relationships between the entities is important. Although the full representation of the data available for a component is captured in the information model, often only a subset is required to be shared with other components; the relationships involving data made available outside of the component are modelled as data views. A view is a projection of attributes within the information model, see Figure 226: Data Views Contain Projections of the Subject Matter Attributes. These are often represented as counterparts, as described in the Component Connections PYRAMID concept (Appendix A: PYRAMID Concepts, section Component Connections).

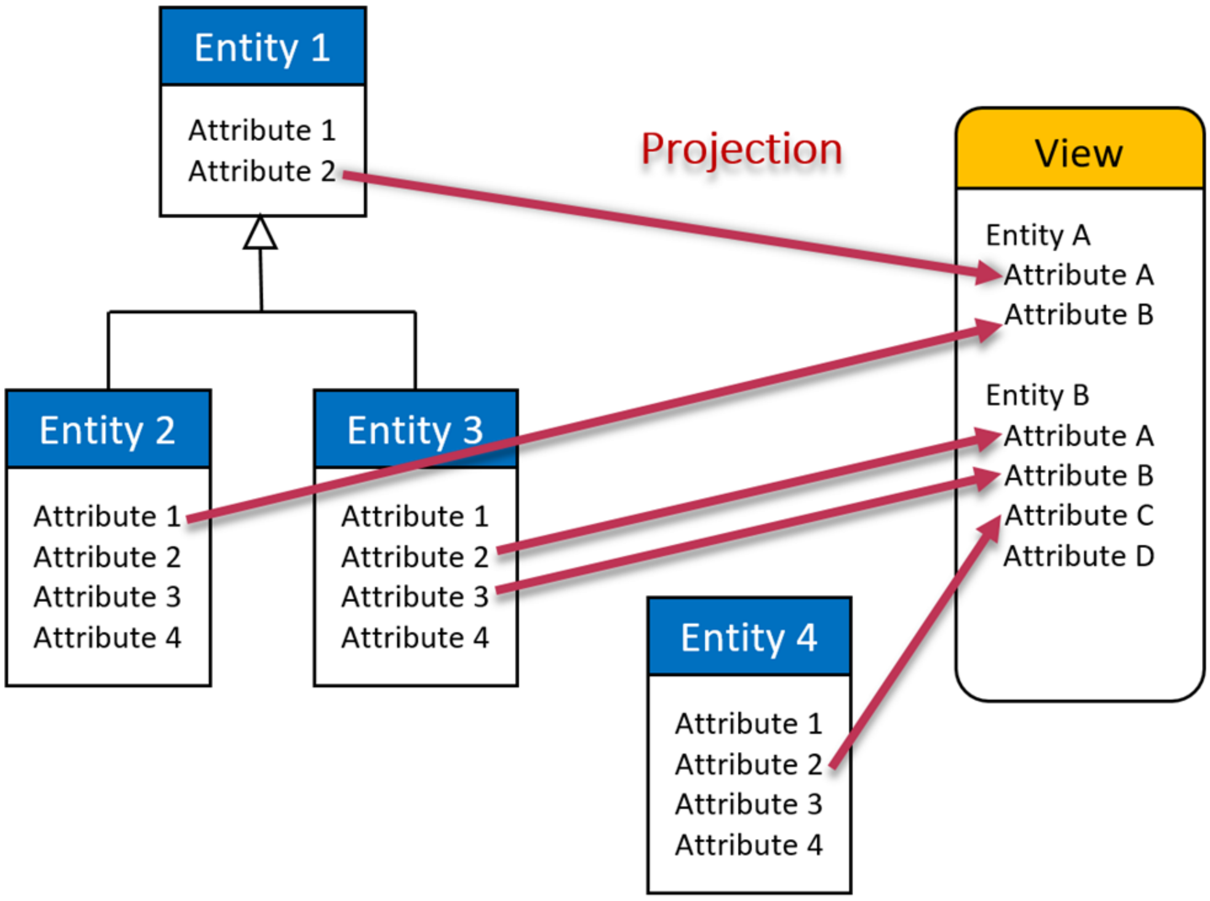


Figure 226: Data Views Contain Projections of the Subject Matter Attributes

System Designers should assess whether an information data model is required, and if so whether it should be a single model or subdivided into relevant sub-models. Where necessary, the management of links and overlaps in multiple data models should also be considered. Only the parts of the information model that are required for the interface specification need to be released to third parties, which is important where Component Developers are concerned with the privacy of their information model.

D.2.9 Good Practice

This section provides good practice recommendations for consideration by Exploiters undertaking Platform Independent System Design, including guidance for avoiding common pitfalls.

Although the recommendations in this section are most relevant to Platform Independent System Design, they may also be relevant to other stages of the design and development of a deployment, in particular Platform Independent Component Design.

D.2.9.1 General Platform Independent System Design Good Practice

D.2.9.1.1 Use Other Standards and Models in Conjunction with the PRA

The PYRAMID Reference Architecture is not designed to contain all the information that is required to design a fully interoperable end system. As a reference architecture, it provides a common component structure and overarching PYRAMID concepts, but other sources of information must also be used in order to allow Exploiters to define a system architecture which meets the requirements for the deployment. Some of these other sources of information may be standards, models and other reference architectures.

In many cases, it will be possible to define a system architecture which is compliant with both the PRA and with any other standards, models or reference architectures which the Exploiter determines to be relevant for the particular deployment. Some of these (including both platform independent and platform specific considerations) are illustrated in Figure 227: Example of multiple sources contributing to the overall system architecture.

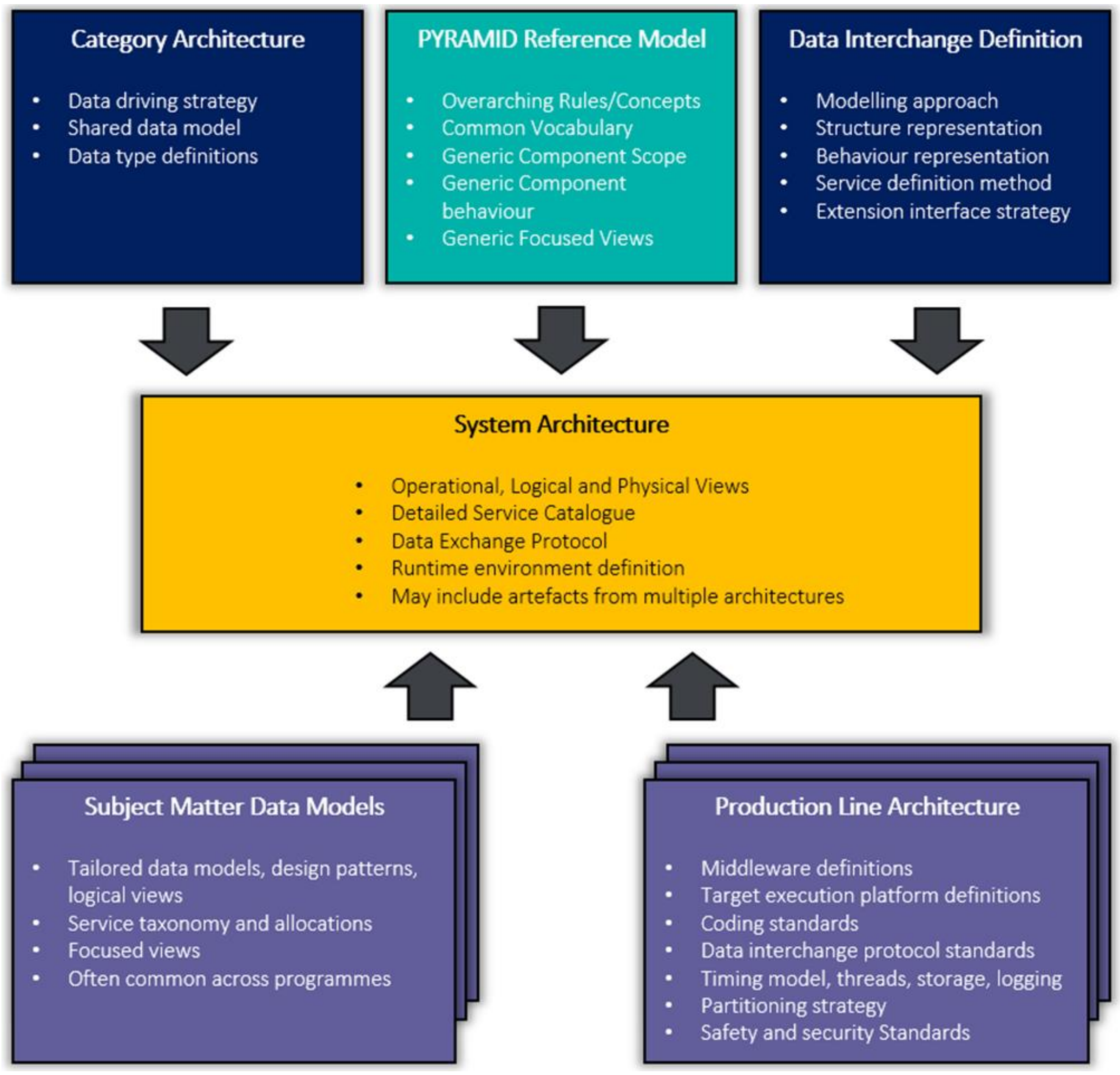

Category Architecture
Data driving strategy
Shared data model
Data type definitions
PYRAMID Reference Model
Overarching Rules/Concepts
Common Vocabulary
Generic Component Scope
Generic Component behaviour
Generic Focused Views
Data Interchange Definition
Modelling approach
Structure representation
Behaviour representation
Service definition method
Extension interface strategy
System Architecture
Operational, Logical and Physical Views
Detailed Service Catalogue
Data Exchange Protocol
Runtime environment definition
May include artefacts from multiple architectures
Subject Matter Data Models
Tailored data models, design patterns, logical views
Service taxonomy and allocations
Focused views
Often common across programmes
Production Line Architecture
Middleware definitions
Target execution platform definitions
Coding standards
Data interchange protocol standards
Timing model, threads, storage, logging
Partitioning strategy
Safety and security Standards

Figure 227: Example of multiple sources contributing to the overall system architecture

This allows a deployment to combine the advantages of the PRA, e.g. the ability to reuse existing components or the production of components which can subsequently be reused, with the advantages of the other standards, models or reference architectures. A common benefit will be interoperability with other systems.

D.2.9.1.2 Delegate Resource Conflict Decisions ‘As Low As Possible’

A principle of the Control Architecture PYRAMID concept (Appendix A: PYRAMID Concepts, section Control Architecture) is that decisions are made by the component that has the appropriate level of information and subject matter knowledge to make the decision. Where conflicts arise, the decisions taken to resolve them should be made as close to the end of derived requirement chains as possible. Patterns which illustrate this can be found in the Dependency Management PYRAMID concept (Appendix A: PYRAMID Concepts, section Dependency Management).

Typically, resolution to a conflict is sought at the lowest level in the control architecture, with components in the Resource Layer prompted to seek alternative solutions to meet a requirement within the constraints or limitations of the requirement. Higher layer components may become involved in resolving conflicts if they are unable to be resolved at the Resource Layer. This may involve re-planning or making arbitration decisions which prioritise one requirement over another. See the Dependency Management and Resource Management PYRAMID concepts (Appendix A: PYRAMID Concepts, sections Dependency Management and Resource Management) for more information.

D.2.9.1.3 Consider which Component is Most Appropriate for Managing Certain Calculations

When a component requires data to be calculated that is not yet available, it is often the responsibility of that component to instigate the calculation of the data when needed. However, a design decision needs to be made concerning which component should actually perform the calculation (and therefore what data needs to be transferred and to where). In some cases the calculation will clearly fall entirely within the ‘subject matter’ covered by one component, and therefore in order to comply with the PRA, the calculation should be performed by this component. However, in some cases the decision may not be as clear cut. In these cases various considerations should be weighed against one another:

- **Keeping the calculation within the component most concerned with the result.** This has the benefits associated with the calculated data being more easily available where it is needed, such as reduced latency or data transfer overheads if the result contains a large amount of data.
- **The amount of data needing to be transferred.** Performing the calculation in a component that contributes a significantly larger data set to the calculation will reduce data transfer overheads and potentially latency in producing the result. Note that:
    - Whilst it may be possible to narrow down the amount of data needed for a calculation within a large data set, additional design or computational effort may be required to achieve this efficiency.
    - Thought should be given to whether current technical limitations that restrict large scale data transfer may be reduced in future.
- **Keeping the calculation in one place.** Where there are common elements across calculations, for example the same algorithm used, it may be beneficial to perform the calculations in the component that contains or is responsible for the common elements. Whilst common software libraries (e.g. mathematical libraries) do enable multiple components to use the same library function without having to redevelop it, there may still be benefits to

restricting the number of components that perform similar calculations, such as easier integration of library changes.

- **Safety and security considerations.** The location of the calculation may need to take account of:
    - The required or desired integrity of the component and/or the hardware that it is intended to be hosted on.
    - The possibility of source or output data corruption, or surveillance during transit, and any additional overheads required to mitigate this.

Assessment may be required to determine where the greatest performance or design effort benefits lie.

For example, a conflict between a proposed route and the terrain would be of most concern to the Routes component rather than the Geography component. However, performing the calculation in the Routes component would likely require large amounts of terrain data to be transferred to the Routes component. If the calculation was performed in the Geography component, the volume of the route data (represented as a line or a simple volume) that would need to be transferred to the Geography component would be much smaller. Furthermore, there would likely be other cases where lines or volumes, representing things in the subject matter of other components, would need to be compared with terrain to determine if they intersect. Therefore, performing the calculation in the Geography component may be the best solution from both a performance and design effort perspective.

D.2.9.1.4 Be Aware of Emergent Properties and Behaviour

Emergence occurs where a group of components have properties and behaviour when interacting as a group that no individual component has on its own.

There are two main problems related to this that should be considered. Firstly, desired emergent behaviour or properties may not be immediately apparent in the component set. Secondly, undesirable emergent behaviour may result from the components being unaware of the behaviour of each other.

The interaction views illustrate one approach to modelling desired emergent behaviour.

Emergent properties resulting from emergent behaviour can often cause confusion, as it may seem that they should be properties within an individual component. One way of identifying these properties is that they can often only be described clearly at an abstract level, covering multiple subject domains. An example is ‘mission context’. Mission context information on, for example, an area of operation may be represented by a set of related data from different components. The entirety of the data may not appear with an ‘area of operation’ label in any individual component. However the concept may well appear at a more abstract level in a higher level component that understands the subject broadly, but with the detailed information distributed between other components that understand those details specifically.

Undesirable emergent behaviour is a problem with any system which is an integration of its constituent parts. However, this is a greater problem with a design for which separation of concerns is the underlying philosophy, such as systems designed to be compliant with the PRA. This is due to the deliberate ‘independence’ of components and emphasis on autonomous behaviour. This may lead to issues such as undesirable ‘loops’ of behaviour, or over processing (e.g. filtering) data in different places which can result in degraded quality and unacceptably extrapolated data. These issues can generally be mitigated through being careful with and understanding the implications of how components interact during design. Careful analysis of alternative scenarios and dynamic constraints can be helpful.

D.2.9.1.5 Avoid Creating a Stove-Piped Design

One of the hallmarks of a legacy style functional decomposition is that it creates ‘stove-pipes’, where lower level resources are under the control of a dedicated higher level component, with all access going through that high level component (see Figure 228: Avoid using multiple copies of the same component in stove-pipes). This leads to a high degree of coupling, in turn leading to code duplication and a tendency for design changes in one component to require changes in other components. This should be avoided in deployments of the PRA.

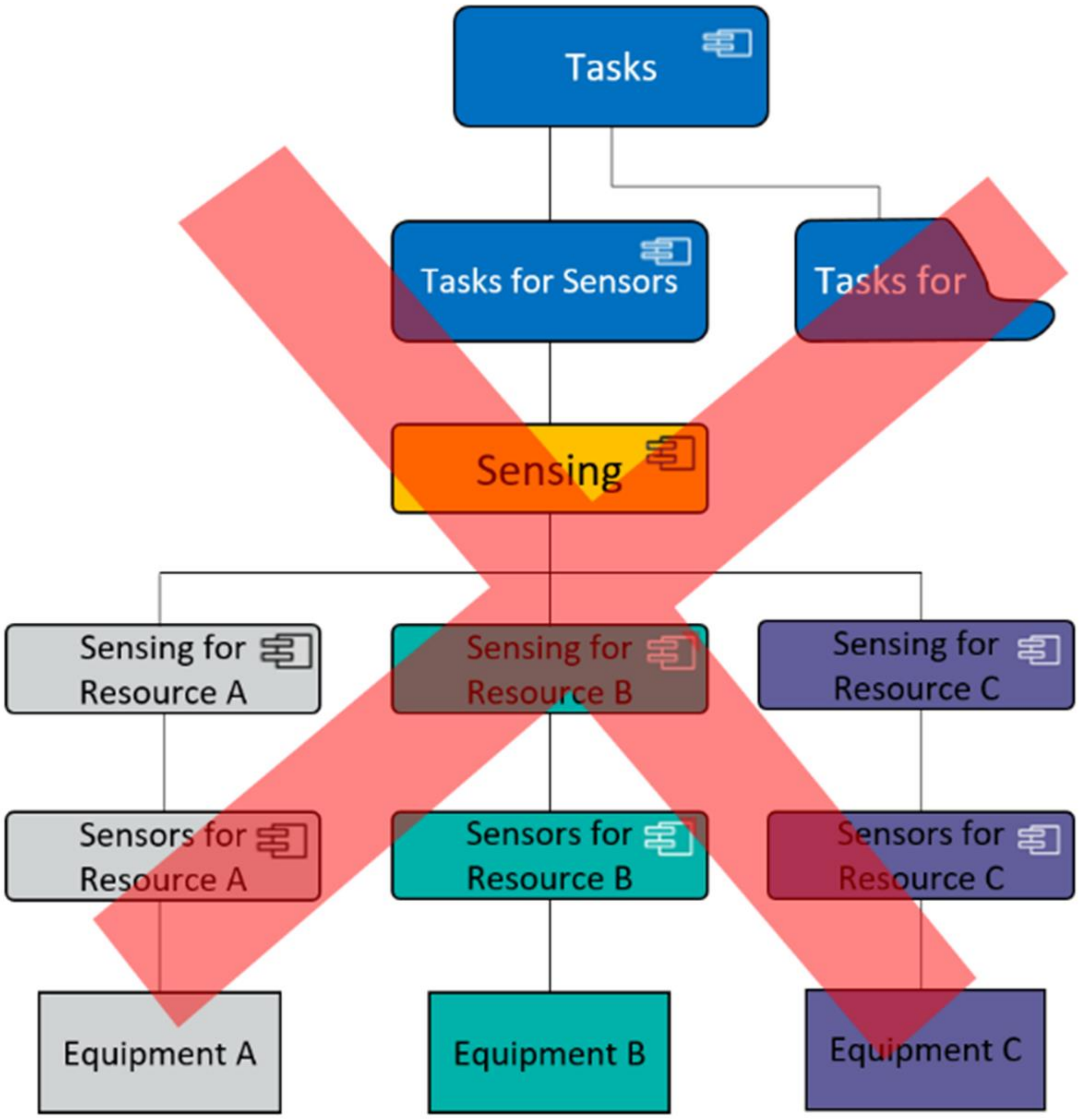


Figure 228: Avoid using multiple copies of the same component in stove-pipes

If a specialisation of a component’s role is needed to support some of the behaviour in new equipment, this should where possible be accommodated using extensions rather than variants. The need for this specialisation should decrease as you go up the abstraction layers in the control architecture: if you need specialisations in a long chain it suggests that abstraction is not taking place, as shown in Figure 228: Avoid using multiple copies of the same component in stove-pipes.

Decisions should be made by a component to which all the required information detail is available, and detailed information should not be available at the higher abstraction levels. For instance, it is reasonable for a Tasks component to decide that to fulfil intelligence tasks a sensing action is needed, and then delegate the details of that to Sensing. As a result there is no place for a dedicated “Tasks for Sensors” specialisation, as all the sensing subject matter knowledge is within Sensing, not Tasks.

Another symptom of this bad practice is the passing of data through an intermediary component to give it to another, as illustrated in Figure 229: Information should be passed directly. Components should access the data they need directly and not via an intermediate component. The control architecture is about the level of abstraction of components, not the control of access to low level data.

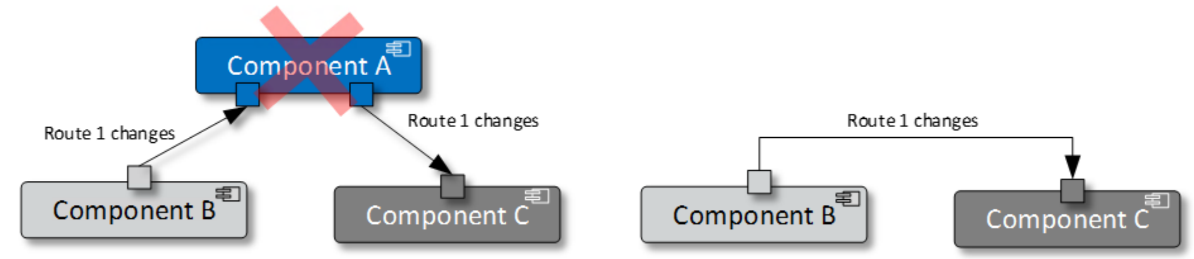


Figure 229: Information should be passed directly

D.2.9.1.6 Avoid Using the Tasks Component as a Stand-In for Planning

A common pitfall when determining the logical separation of the system is to only consider certain stages in the mission lifecycle, for example by only considering the active stage of a mission with regards to the Action Layer components, and think that another component (e.g. Tasks) will provide the planning. Tasks will provide the requirements for an activity, but this is about high level abstraction, not about determining exactly what should be done prior to a mission taking place. Separate variants or instances of components might be used for planning, as shown in Figure 230: Tasks is not a stand-in for planning. It should not all take place within an instance of the Tasks component. If data from an Action Layer component is being duplicated in the Tasks component (or an extension of the Tasks component), then this suggests that the abstraction from the lower level actions in the Action Layer components to the higher level tasks has not been properly understood.

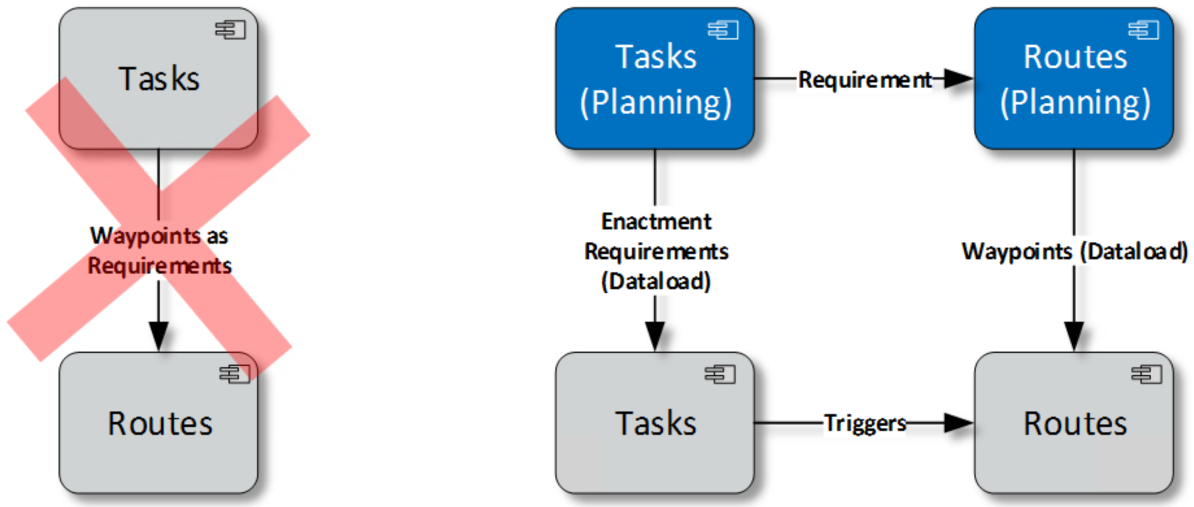


Figure 230: Tasks is not a stand-in for planning

D.2.9.1.7 Avoid Treating the Tasks Component as a Supervisor

A common problem is to think of the control architecture as a control hierarchy with Tasks sitting on the top as a substitute for a pilot in a supervisory role. A symptom of this is long chains of status messages being passed between components, triggered by the same event, with Tasks acting as a progress tracker even for ‘lower-level’ activities.

The control architecture is about abstracted views of activities; each component should only report progress at the level of its own understanding. Reporting of ‘low-level’ progress milestones to ‘higher-level’ components is unnecessary, as any problems and decisions should be handled by the component that has enough information detail to correct the problem. Tasks will only be involved in the abstracted view of the activities, as illustrated in Figure 231: Reporting between components.

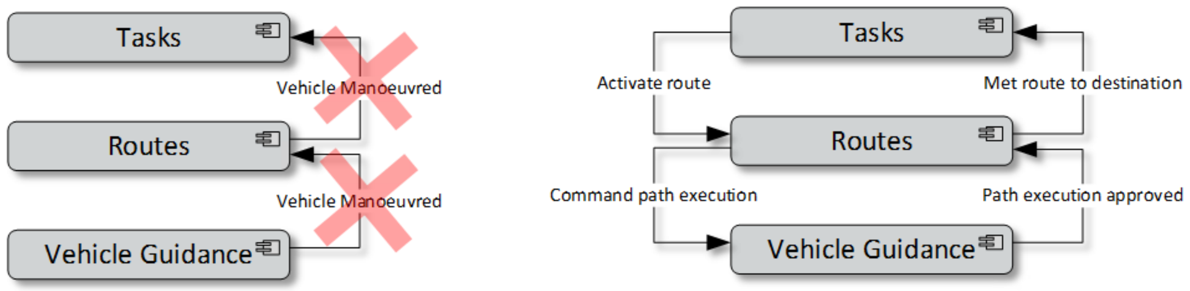


Figure 231: Reporting between components

Tasks should not need information about the progress of a component at the level of detail understood by that component, but if it does, it can go directly to that component for the information. It does not need to be passed up a chain. Similarly, if a user wishes to view low level information about an activity this would be provided to the HMI directly from the related component.

D.2.9.1.8 Avoid Over-Defining Inter-Component Behaviour during System Design

When looking at the interactions between components that need to take place in order to enable a use case, specifying the interactions in too much detail can lead to the component specification overly constraining the component design to do things in one particular way, which can make it harder to incorporate a change into the system at a later date. An example of this is specifying detailed handshaking protocols between components.

Likewise, although the component specification will need to specify the basic internal logic that is required from the component, too much focus on multi-domain ‘sunny day’ and ‘rainy day’ scenarios may drive a component’s design to be overly specific and less resilient to change or reuse. A potentially better approach to ensuring a component is resilient to ‘rainy day’ scenarios is to consider simple ‘rainy day’ interaction patterns for the individual component, for example by considering what happens when a request made to another component is not carried out. This more general approach can give the component a more generalised resilience to error conditions, rather than one that is specifically tailored to particular scenarios.

D.2.9.1.9 Consider Coordination between Nodes

The components can be designed to cover the numerous types of situation or setting that the deployment may be involved in (e.g. training, maintenance support, and from mission planning to post mission data analysis). At the platform independent stage of design it is assumed that every component has access to the data that it needs to carry out its role, however, there needs to be consideration of how to coordinate data between multiple locations. Data coordination requirements are likely to drive out specific services and dependencies to be captured in the service contracts and service dependencies. For further guidance on coordination between vehicles in operation see the Multi-Vehicle Coordination PYRAMID concept (Appendix A: PYRAMID Concepts, section Multi-Vehicle Coordination), although note that this is only one specific case out of many types of coordination between nodes.

D.2.9.1.10 Consider Interaction with Humans

Specific components are provided for interaction with humans (see Appendix A: PYRAMID Concepts, section Human-Machine Interface). However these HMI components do not have any ‘domain’ knowledge, so to meet human interaction needs other components will be involved via the HMI components, and the implications of this should be considered. For example, needs to present information, such as a communications heat map, would require additional data to be held within an appropriate component.

D.2.9.1.11 Consider Interaction with Equipment

Components are needed to determine the capability of equipment or parts of the software infrastructure, and to understand how to control hardware resources. For guidance on developing these components see the Interaction with Equipment PYRAMID concept and Interfacing with Deployable Assets PYRAMID concept (see Appendix A: PYRAMID Concepts, sections Interaction with Equipment and Interfacing with Deployable Assets). Some guidance on the boundary of the PRA with respect to equipment hardware is also given in the Smart Equipment and the PRA/Equipment System Boundary subsection of the Interaction with Equipment PYRAMID concept (Appendix A: PYRAMID Concepts, section Interaction with Equipment). Interface requirements with existing equipment also need to be taken into account.

D.2.9.1.12 Use Counterparts

The principle of components dealing only with data within their own subject matter knowledge means that different components are likely to have their own views on the same real world ‘thing’ or concept, such as some liquid fuel on an Exploiting Platform. A Mass and Balance component may hold data on fuel mass, whereas a Fluids component may be concerned with fuel volume distinct from and regardless of fuel mass. Meanwhile an Environmental Conditioning component may view the fuel as a heat source or sink so may hold data on its thermal properties. In such a case each component may contain a class to represent the fuel from the component’s subject matter specific point of view, with these classes and their instances being used to form a counterpart relationship. See the Component Connections PYRAMID concept (Appendix A: PYRAMID Concepts, section Component Connections) and Ref. [6] for more details on counterparts.

Counterpart relationships support separation of information between the relevant domains, helping avoid subject matter pollution. Key benefits of such an approach are summarised below:

- The domain based approach to components is predicated around minimum amounts of information being passed between components because each component should handle information within its own subject matter. However, it is normal in an integrated system that information relating to different aspects of the same real-world object or concept will need to be handled by different components. Identifying counterparts and using counterpart relationships between these different items of information allows traceability through the system.

- Information on a real world object or concept known to the deployment can be obtained efficiently, with relatively low amounts of data passing between components, for example through the use of system wide identifiers to obtain data about the relevant counterpart from the owning components. This also applies, in a broader sense, to management of information about counterparts, potentially via standard operations.
- By mapping counterparts (via bridges), their data can be synchronised between domains to enable coordination and consistency across a deployment.
- Use of counterparts supports the extensibility and scalability of a deployment, as it allows easy addition or removal of information types made up of data linked through counterpart relationships. It enables specialisation of shared interfaces between domains, regardless of programming language and execution platform.
- Counterparting is a suitable way to enable data views (see the Component Information Data Model and Data Views section), since counterpart relationships can indicate the attributes that a component may need to make available to other components.

Note that while a counterparting approach is effective for capturing relationships between the information in different subject matter areas, there are some event-based relationships, such as command and control, where the counterparting approach may not be appropriate.

D.2.9.2 Service-Specific Good Practice

D.2.9.2.1 Follow S.O.L.I.D. Guidance

S.O.L.I.D is an acronym for five of the key object-oriented design principles first put forward by Robert C. Martin in Ref. [24], which should be considered when designing both services and components. Some of these principles will be more relevant to the Platform Independent Component Design, but they are included here as service definition is the stage in the development of a deployment where they first become relevant.

The S.O.L.I.D. principles are:

- **Single-responsibility principle** - A service should have one job, giving it one reason for change. This principle also extends to classes within a component. This segregation will help the design in many areas, from partitioning the software to safety case creation.
- **Open-closed principle** - Components (and all other parts of the design) should be open for extension, but closed to modification. This means that additional functionality should be able to be added to them, but existing functionality should not be modified. A key aim for a PYRAMID deployment is to be open to change while still conforming to safety requirements applicable to its operational environment. This single principle can contribute significantly towards meeting this aim.
- **Liskov substitution principle** - Extension components should add to (or extend) a parent component's behaviour, not replace it. This principle should also be considered in relation to polymorphism within the component.

- **Interface segregation principle** - A component (or bridge) should never be forced to implement an interface that it doesn't use, either as a consumer or provider of that interface.
    - As there would never be a need to implement an entirely unused interface, this principle is more about the interface’s ‘size’ - if a component providing a service defines an interface as providing a large block of data, while the consuming component needs only a small amount of that data, then the ‘other end’ of the interface needs to implement functionality to receive all of provided data, only to immediately discard most of it. Segregating the interface into a number of smaller, more specific interfaces means that the other end of the interface only has to implement functionality to receive the data that it needs, and removes an unnecessary dependency on the parts of the interface that aren’t needed.
    - In a PYRAMID deployment, each component’s interfaces are specified independently, and bridges are used to connect components. This means that it is bridges that would be forced to implement ‘unused interfaces’ if the component interfaces were not well defined, and bridges that would have the resultant unnecessary dependencies on components.
- **Dependency inversion principle** - Components should depend on abstractions, not on concretions. In addition, abstractions should not depend on details, but details can depend on abstractions. This is about decoupling in component design, and should come from applying the open-closed and Liskov substitution principles. This principle should also be applied within a component.

Other reference texts with guidance for object-oriented design practice are widely available. For example, Ref. [25] provides general guidance on patterns and anti-patterns in object-oriented modelling of systems, whereas Ref. [22] provides specific guidance on object-oriented design in avionic systems.

D.2.9.2.2 Design Highly Cohesive and Loosely Coupled Services

The PRA components have been designed to each have a well-defined, tightly related subject matter. Designing highly cohesive components this way leads to the relationship between components being loosely coupled. This principle should be extended to the services. Services should be designed from a consumer focus, but also from the component’s perspective, i.e. consider a generic consumer for the component, but also consider the needs of the specific component under design.

Service design should be approached with care by those who are new to Service Oriented Architecture (SOA), with specific attention to considering services from the consumer focus and understanding the difference between services provided by the component and those which are consumed by the component. Those new to SOA may be inclined to consider services as exchange of data, rather than requests for work to be done, which can result in poor service design. For more information on SOA see Ref. [7] and [8].

D.2.9.2.3 Design Reusable Services

The future of technology is hard to predict. Even if a service is only used by a single other component in one deployment, it should still be designed so that it can be reused in the future, when the component may be used in a completely different context in another deployment.

However, when designing services that you expect to be reused in various ways, care needs to be taken to not define a parameter or attribute that is so vague that it is meaningless. Quality and capability measures are particularly prone to this.

D.2.9.2.4 Design Services that are Decoupled from Internal Logic

While the best component services are often developed in collaboration with Component Developers, the interface should be abstracted from how the service is fulfilled. This allows for changes to the internal structure of the component to take place without impacting the interface, and vice-versa.

D.2.9.2.5 Design Composable Services

Services can be composed of other services, which allows capabilities to be orchestrated at different levels of granularity. Composition in services can be promoted through common service patterns. General guidance on service patterns can be found in the Component Connections PYRAMID concept (Appendix A: PYRAMID Concepts, section Component Connections).

D.2.9.2.6 Design Independent Services

To facilitate simple choreography, services should be designed with the ability to be independent, having self-control over the logic they govern within an explicit boundary. Therefore, services should be designed to be able to stand alone from other services with the minimum of conversation between the consumer and provider (see Ref. [5]). This allows for a greater decoupling between components, and as a result greater independence in the development of components. If there are many one-way flow services with a high degree of dependency between them, this is a symptom that this has not been considered.

D.2.9.2.7 Design State Independent Services

Services should minimise the amount of state information managed on behalf of consumers (users of the service). When a single transaction for a given consumer involves multiple service calls, there is sometimes a need for a service to manage the state for its consumer for the duration of the transaction. This need is greater in loosely coupled components, as state is private to the component providing the service. This maintains high cohesion and keeps consumer interactions simple, but is inefficient, scales poorly and is less resilient. To avoid these problems, services expect the transaction state to be maintained by the consumer. The consumer passes the transaction state into each service call related to the same transaction.

In the PYRAMID modelling approach, this transaction state can be maintained in a counterpart class instance that holds a relationship with the corresponding counterpart instance existing from the provider component's perspective. The consumer component's counterpart class is a representation of its understanding of the transaction. This counterpart's data will then be passed in any service calls and translated by the bridge into terms understood by the component providing the service. Following this approach will reduce resource consumption and enable the service to efficiently handle multiple concurrent consumer requests.

Symptoms of this principle not being followed are the need to capture states and mode information in bridges (making them over complex and potentially polluted with functionality that is within the scope of a PRA component), or having to pollute another component with knowledge of the component in question's state. Over-complex bridges that have a high Development Assurance Level (DAL) potentially make certification more difficult, among other issues.

Often, to achieve state independent and atomic services it is necessary to pass more information in the service operations in order to avoid reliance on state management between calls, which leads to more complex service operations.

D.2.10 Summary

System level artefacts that should be identified by this stage are:

- **Build sets** that define which components are required to meet the customer requirements; see the Architecture Definition section.
- **Component specifications,** which are populated with the parameters to be exchanged; see the Component Specification section.
- **Bridges**, which are populated with mappings that connect the services of interacting components; see the Bridges section.

D.3 Platform Independent Component Design

D.3.1 Introduction to Platform Independent Component Design

A Platform Independent Component Design defines a component’s internal behaviour, in order to show how it should satisfy the component specification and thus fulfil its responsibilities within the overall deployment. This section covers both how the PRA supports the Platform Independent Component Design stage for a component, and other considerations which should be taken into account in order to design a component that supports achievement of the PYRAMID benefits.

Loose coupling between components is a key feature of the PRA, and therefore the Platform Independent Component Design stage should take place independently for each component.

The specification on which the Platform Independent Component Design should be based is produced as part of the Platform Independent System Design. This defines the purpose of the component, in terms of what it provides and what it needs from the system. See the Component Specification section for more details.

D.3.1.1 Updates to the Component Specification

As the component design is developed, the Component Designer may discover the need for new services (in particular consumed services) in order to enable the required behaviour. If this happens, they should consult with the System Designer to agree any changes to the component specification. These changes will form a new iteration of the Platform Independent System Design. This will help to ensure that the Platform Independent System Design and Platform Independent Component Design remain consistent.

D.3.1.2 Reuse of Existing Components

Where an implemented component already exists that meets the component specification (or allows the component specification to be met when the component is correctly configured), and fulfils any other needs of the deployment, reuse of this component should be considered. It may also be useful to consider whether an existing component could be adapted by the development of a new extension, in the case where the component has existing extension points (see the Identification of Extensions section and Appendix A: PYRAMID Concepts, section Component Extensions, for more details).

In cases where an existing component is being reused, in some cases much of this section will not be required. However, in cases where the component needs to be correctly configured to meet the specification, for example through data driving, a Platform Independent Component Design will still be required to define the internal behaviour that should be provided by the component configuration.

Where a deployment aims to reuse PYRAMID compliant components, due consideration must be given to the potential impact of changes to PRA component definitions between different versions of the PRA (see section, Different Versions of the PRA).

D.3.2 Component Requirements

The requirements placed on the components by the system design, which specify the functionality required from each component, may need to be analysed to support the further maturation of component design artefacts. As with the system level, the requirements need to be analysed through a normal system engineering process (see Ref. [5]). At the component level, this section assumes a separation of concerns from non-functional requirements concerned with how the software runs on the execution platform, addressed in the Platform Specific System Development and Platform Specific Component Development sections.

D.3.3 Component Modelling

The modelling performed by the System Designer will have defined the component specification, which identifies the component's services along with the data required to support them (including data-types for parameters referenced by the interfaces). Future systems are required to be rapidly adaptable at an increased frequency with an overall reduction in through life costs. This section covers what component modelling techniques can be used to design a component to meet its specification while supporting this goal, and what assistance is provided in the PYRAMID Technical Standard, Ref. [1] and PYRAMID Technical Standard Guidance document.

D.3.3.1 Structural Analysis from Data

The PIM focuses on modelling high-level concepts to achieve the highest possible level of reusability, as the level of reusability inversely correlates to the amount of low level detail specific to the particular deployment which has had to be included. Object-oriented design techniques should be used to define the system, for which additional guidance is given in Ref. [21] and Ref. [22].

A system wide data model, which could apply to the whole system (e.g. the air vehicle) or just parts of the system (e.g. the mission system), can be used to help identify what data is available at the component interface. However, this does not provide a detailed structure of what the component needs to hold internally, which needs to be considered further to identify software structure. Typically this information is captured as a class diagram, and should address the following:

- Consider the “items” (the actualisation of the entities) that the component reasons about and manages. The semantics diagram within a PRA component definition can be used as a starting point for this by providing context to the component's subject matter entities. The component specification will have defined the public entities, i.e. those entities needed to support the services. Those entities, within the subject matter part of the component specifications, can be considered as a conceptual definition for the “items”.
- Consider whether any additional internal entities are needed to support the public entities. The Component Designer should address this to enable the component model to be completed, and should also take into account the component's scope boundary (as defined in the PYRAMID Technical Standard, Ref. [1], section Component Definitions).
- Consider the information and state data that entities need to capture. This is done by defining attributes on the classes to capture the data they own, or data they use to capture their current state.
- Consider the relationships between the entities. This defines which entities make use of or reference other entities. Note that references between entities are implied attributes - so there is no need to model attributes to show containment or aggregation of entities.
- Consider the operations needed to support the required behaviour. This includes operations needed to support services and any autonomous/internal behaviour.

The structural and behavioural design of the component following standard object-oriented design practices may drive out additions to the data model to support data driving. Examples include configuration data such as maximum and minimum limits of allowable values for a particular class. Methods for loading this configuration data should also be considered when modelling the internal structure.

D.3.3.2 Data Driving

PYRAMID promotes a data-driven approach to the development of the Platform Independent Model (PIM). Appendix A: PYRAMID Concepts, section Data Driving, explains what data driving is, its benefits in supporting component reuse and rapid adaptability, and how it can be realised in practise. A brief outline is provided here, but the reader should refer to this PYRAMID concept for further information.

The process of data driving involves controlling component behaviour through external configuration data, defined through separate data sets. Data driving allows defined data sets to be applied to the creation or execution of executable software to modify component behaviour. If applied at build time, the data forms part of the compiled executable software component and may be used to create unique component variants. If applied at runtime, the data is acted on by the executable and may be used for example to support variations that occur in operational use, such as different platform configurations or mission types.

At the PIM phase of development consideration needs to be given to the nature of the variability that is required in component behaviour, and whether this variability is required to support the development of unique component variants or to drive variability at runtime. The resulting PIM design then needs to make provision for the different data-driven aspects. A structured approach to the use of data driving can often uncover a wide range of opportunity for variability that might not otherwise have been identified, thus supporting the goals of reuse and rapid adaptability. Appendix A: PYRAMID Concepts, section Data Driving, lists the considerations for Exploiting Programmes when looking to apply data driving.

D.3.3.3 Behaviour Modelling

Static structural diagrams (e.g. class diagrams and entity relationship diagrams) only tell part of the story. It is also important to capture the behaviour expected from services. This is done using activity and sequence diagrams showing the relationship between external services and internal operations.

State transition diagrams should also be considered for classes that manage state data, showing how they will behave for different situations and operations. This can also help drive out additional required information. The level of detail to be modelled depends on how formal this needs to be. For safety critical systems this may well include action language statements providing a clear definition of the expected behaviour.

Use case diagrams are also important, as they help to represent the functionality of the component under analysis in terms of how it is intended to be used by external actors. Use case analysis is recommended as it is a modelling activity that is customer-focused but component-centred, and can capture the intended behaviour of individual components.

With this information in place it is then possible to design the test cases that will be used in testing of the component prior to integration.

D.3.4 Design Areas of Concern

This section outlines a number of considerations which should be taken into account when producing the Platform Independent Component Design.

These should be considered alongside:

- The design considerations and assumptions contained within the PRA component definition for the particular component under design. These can be found in the PYRAMID Technical Standard, Ref. [1], section Component Definitions.
- The PYRAMID concepts, provided in Appendix A: PYRAMID Concepts. The PRA component definitions provide a suggestion of which PYRAMID concepts might be particularly relevant to a component, but this is not intended to exclude consideration of other PYRAMID concepts.

There are significant overlaps between the PYRAMID concepts and the design areas of concern included below, and references are made to the relevant PYRAMID concepts where appropriate.

D.3.4.1 Data Configuration

Identifying which types of data within the component should be data-driven can take place before the internal structure of the component is determined and without restricting the internal structure of the component to any one design style.

Allowing a component to be configured by data-driving is beneficial because it means the same implemented component can be used within different deployments (for example within different Exploiting Platforms or to support different hardware setups). It can also allow adaptation to different end-user scenarios (for example different role-fit or operational requirements).

However, adding configurability is also likely to increase the complexity of the verification process for the component and any deployment containing the component, and associated cost implications should be considered when deciding whether to make a component configurable. For further guidance see the Data Driving PYRAMID concept (Appendix A: PYRAMID Concepts, section Data Driving).

D.3.4.2 Autonomy

In order to aid understanding of the likely behaviour of a component, the PRA components have been grouped by levels of expected decision making responsibility and control. For further guidance see the Control Architecture PYRAMID concept (Appendix A: PYRAMID Concepts, section Control Architecture).

Whatever the level of the component within the architecture, if it has the information needed to make an informed decision then it should make the decision rather than delegate it or push it up the chain to another component. This information may include the authorisation to make the decision. The level of autonomy that is supported, including requirements for when human interaction is needed, should be captured as part of a service contract. For further guidance see the Autonomy and Human-Machine

Interface PYRAMID concepts (see Appendix A: PYRAMID Concepts, sections Autonomy and Human-Machine Interface).

D.3.4.3 Constraints

To support its decision-making, a component needs to provide the facility for the correct determination of constraints. Constraints may be provided as part of configuration data (see the Data Configuration section) and loaded during initialisation, through planning activities, live via other mechanisms, or as an integral part of a component's data model. Component Developers should provide sufficient information to aid initial component integration, such as example constraint data. This will allow the System Integrator to better understand what is expected from the supporting components and infrastructure. For further guidance see the Constraint Management PYRAMID concept (Appendix A: PYRAMID Concepts, section Constraint Management).

D.3.4.4 Capability and Health

It is important that, where appropriate, components are able to assess their capability over time and with use, including the ability to predict capability progression, and to highlight missing information that could improve the certainty or specificity of the assessment of their capability. This capability assessment functionality can support activities for alerting a user to anomalies, offering remedial action or performing automatic corrections.

Components should be able to identify how loss of capability that they depend on impacts their own capability. Documenting service dependencies, typically in a service dependencies diagram, could help indicate implications of failure in the chain.

As well as Anomaly Detection and Health Assessment components monitoring the health of the underlying infrastructure that a component relies on and is aware of, the component needs to be able to provide information for determining whether there are anomalies that are the result of systemic changes, which may be the result of faulty or corrupted software.

A Component Developer should provide data to support the configuration of health and cyber defence monitoring components (including detection of anomalies), allowing other components to better characterise changes in the capabilities of the system. For further guidance see the Appendix A: PYRAMID Concepts, sections Capability Management, Health Management and Cyber Defence.

D.3.4.5 Safety Related Dependencies and Rely-Guarantees

The Service Dependencies section covered definition of dependencies between services within the service specification. Now that the logical separation within the component is better understood, more detail can be added to this relationship. Rely-guarantees can be used to express the relationships between the services that a component guarantees and the dependencies it relies upon to provide those guarantees. A rely-guarantee method allows the behaviour of services to be abstractly specified, meaning that they can be verified separately without requiring extensive implementation details.

Where dependencies are expected to affect safety aspects of the system, rely-guarantees can be used to support modular certification. Safety information can be overlaid onto rely-guarantees to provide a safety view concerning the relationships in question, and they can be expressed at the desired level of component abstraction to support certification at that specific level. They can be refined for decomposed lower levels of design, down to the effect of individual code segments.

Goal Structuring Notation (GSN) may be utilised in conjunction with rely-guarantees to support safety certification (see Ref. [23]). GSN can be used to provide a reasoned argument that component guarantees will be met and provide the supporting evidence.

D.3.5 Identification of Extensions

In order to maximise reuse and reduce integration and testing overheads, the PRA allows for extension components, which can be separately modified, added or removed when needed. For an explanation of extension components and how these relate to the PRA, see the Component Extensions PYRAMID concept (Appendix A: PYRAMID Concepts, section Component Extensions).

The designer should consider whether the component's behaviour benefits from being factored out into extensions. The System Integrator should be consulted about the areas of functionality that are likely to change through the system's life. This will help identify where use of extensions could reduce time and effort compared to that required when changing the whole component, due to the additional functional separation that they enable.

D.3.6 Good Practice

Although the recommendations in this section are most relevant to Platform Independent Component Design, they may also be relevant to other stages of the design and development of a deployment.

D.3.6.1 Avoid Pollution

Pollution is where a component has been expanded to the point that its subject matter now overlaps with the subject matter of another component. Pollution may result in a PYRAMID non-compliant component as described in Appendix E: Compliance Guide. (Even if the pollution does not affect the component's provided services, meaning the component is technically compliant, including pollution within components is against the ethos of the PRA and is likely to negatively impact the achievement of the PYRAMID benefits).

The following points explain how to avoid common occurrences of pollution:

- Do not force details into a component that understands a subject at a higher level of abstraction so that it can "manage" the components that understand the specific detail. Components that understand a subject at a higher level of abstraction should not know details.
- Do not duplicate data in one component that belongs to another component.
    - Use the data in the component without duplicating it - how this data will be accessed is a concern for the platform specific modelling.
    - One way of hiding the fact that data is being copied is by renaming the data classes - ensure that this is not occurring.
    - The components have been designed to be loosely coupled. If you need to duplicate data, checks should be made that you are not also duplicating functionality.

- Do not be too implementation specific in components higher in the control architecture. Producing versions of these components that are highly specific to the needs of the deployment may make them simpler, but will reduce their reusability. They should stay at their designed level of abstraction to avoid pollution and promote reusability.

D.3.6.2 Follow S.O.L.I.D. Guidance

The guidance contained in the Follow S.O.L.I.D. Guidance sub-section of the Platform Independent System Design section may also be relevant here.

D.3.6.3 Consider Mission Planning

Planning and execution of activities are not fundamentally different when it comes to the subject matter that the system must reason about. The following guidance is recommended:

- Do not treat mission planning as solely an off-board process, all execution involves some level of planning.
- The parts of a plan that are either requirements or solution options should be clearly and distinctly defined as such. This enables design that can allow plans to be changed appropriately as circumstances change.
- Do not assume that smart equipment is unable to support extensive re-planning on the fly.

Note that maintaining the level of abstraction of components will promote reusability, as components will support both planning and in mission use.

D.3.7 Summary

PIM artefacts that should be identified by this stage are:

- **Classes** which represent the subject matter of each component, depicted on class diagrams which define the relationships between these classes. These classes should contain the attributes and operations required to fulfil the component specification, including supporting the services of each component.
- A **data driving** model, representing any data driving requirements.
- **States** for classes whose behaviour is state-dependent (usually depicted using state diagrams) which represent the sequences and cycles of states entered by instances of the class as they progress through their lifecycle. Note that state here refers to states which control behaviour or lifecycle, and not simple changes to data values.
- **Extensions** for some components, including services for interaction with the parent.

D.4 Platform Specific System Development

D.4.1 Introduction to Platform Specific System Development

The Platform Specific System Design stage considers non-functional requirements and infrastructure architecture strategies for the overall deployment, while still remaining independent of the detailed mechanisms of the underlying middleware or operating system functionality that is to be used to achieve the desired result. It adds detail to the Platform Independent System Design and further defines the design of the overall deployment.

The infrastructure, or execution platform, design needs to be understood so that it can be related to the Platform Independent System Design outlined in the Platform Independent System Design section. This section will focus on the impact of the execution platform design on the components and on the Platform Independent System Design.

The addition of this infrastructure detail to the Platform Independent Model (PIM) results in the formation of the Platform Specific Model (PSM). Although it is possible to miss out the modelling of the PSM and go straight to the application software code, including this step supports the underlying principle of *separation of concerns*. This allows for solutions to problems to be considered in a narrow context, avoiding overcomplicating parts of the design stages and missing the impact of some of the decisions.

D.4.2 Non-Functional Requirements

The non-functional requirements considered during this stage should be those that are concerned with how the system design as expressed in the Platform Independent Model is implemented on the infrastructure which forms the execution platform, and how this affects the development of the application level software. For example, this might include requirements related to:

- **Safety** - there may be a need for multiple instances of components on different elements of the infrastructure to provide redundancy.
- **Security** - there may be a need for multiple instances of components on different elements of the infrastructure to allow for separation of information of different classifications.
- **Performance** - in order to achieve required performance, the processing speed provided by the infrastructure may need to be considered alongside any particular algorithms which have been chosen.

D.4.3 Infrastructure Architecture

The Exploiting Programme needs to make a number of decisions about the specific infrastructure, or execution platform, that will support the components that form the PYRAMID deployment. These decisions are outside the scope of the approach to development of a PYRAMID deployment described in this appendix, but will be based on a number of factors, including the scale of the deployment and whether the deployment needs to be distributed across a number of computing resources, for example in cases where the Exploiting Platform includes multiple nodes. The infrastructure that supports the components is typically a software infrastructure including middleware, operating system and hardware (see the Layered System Infrastructure section), but depending on the implementation it may include Field-Programmable Gate Arrays (FPGAs), crystal oscillators, integrated circuits or relays.

Platform Specific System Development includes the transformation of the Platform Independent Model (PIM) into a Platform Specific Model (PSM), and the complexity of this transformation will depend on the complexity of the execution platform. For example, if the infrastructure needs to support deployment of the components across a distributed computing architecture then that is more complex than deploying to a single piece of executable software running on a single computer.

This section outlines some specific infrastructure-related areas that need to be considered at this stage of the design.

D.4.3.1 Layered System Infrastructure

A layered infrastructure architecture is recommended for a deployment where possible, as this allows the application software to be decoupled from the underlying infrastructure. This layered approach employs a well-defined interface between the applications and the infrastructure. The number of layers in the infrastructure is dependent on the precise implementation. Figure 232: Example Deployment Architecture illustrates an example layered approach.

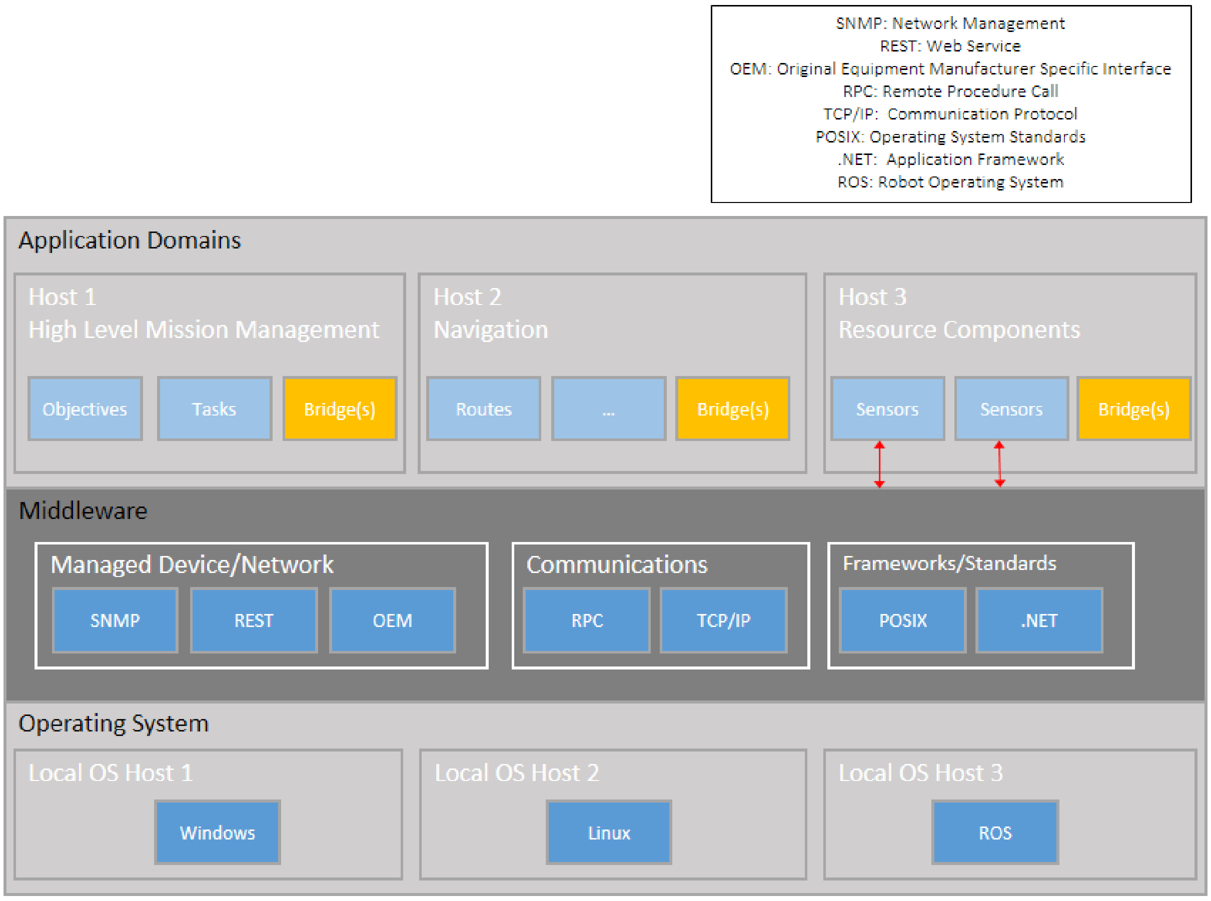


Figure 232: Example Deployment Architecture

Communication between components should use bridges to connect their defined services. However, a component is allowed to access middleware directly for specific purposes within its subject matter, such as:

- Reading the system clock;
- Controlling or interrogating a hardware resource;
- Inspecting health data.

Access to middleware must never be:

- Beyond the scope of the component's subject matter;
- Instead of using the component's defined interfaces.

It is expected that an Exploiting Programme will use an industry standard Application Programming Interface (API) for access to middleware. For further guidance on a layered architecture see the FACE® Technical Standard, Ref. [20].

D.4.3.2 Support for Scheduling

An infrastructure architecture for a deployment must consider a run-time system that is capable of scheduling. This includes not only scheduling of the software items but also threads within those (i.e. support to multi-threaded software items). Different types, extensions and combinations of scheduling algorithms should be considered (e.g. earliest deadline first or shortest job first) when choosing the work scheduling method. Scheduling of access to managed resources is detailed in more depth in the Resource Management PYRAMID concept (Appendix A: PYRAMID Concepts, section Resource Management).

The software infrastructure must also provide a mechanism for implementing threads and semaphores for protection of multi-threaded code. For more information on model based design for real-time systems and the importance of thread management see chapter 8 of Ref. [5].

D.4.3.3 Software Resource Allocation and Protection

The expectation is that some parts of the deployment system will be safety critical, and so the software infrastructure will need to support safety critical software. In each case, the infrastructure architecture must consider the safety criticality of the software it supports.

A PYRAMID deployment needs to consider software resource allocation, and protection between software items or between groups of applications (protection domains). Software items should not be capable of accessing the memory space of other software items.

All communications between applications (except those in the same protection domain) should go via the software infrastructure. For lower safety criticality levels, dynamic memory allocation should be possible, and this should be constrained to avoid interference between applications.

D.4.3.4 Time

An Exploiting Programme may have a requirement for a common time reference across system nodes (e.g. multiple air vehicles or items of hardware within an air vehicle). When choosing a time synchronisation method, the system design should take into account its use of the infrastructure as a time source.

D.4.3.5 File Access

A PYRAMID deployment must consider a file access mechanism for software items, which should allow both read and write access to files, and should be agnostic of the hardware mechanism for data storage. These requirements apply whether data storage is provided on each line replaceable unit or centralised within the system. For additional guidance see:

- The Recording and Logging PYRAMID concept (Appendix A: PYRAMID Concepts, section Recording and Logging);
- The Storage PYRAMID concept (Appendix A: PYRAMID Concepts, section Storage);
- The Cryptographic Management interaction view - for protection of files accessed by different applications (Appendix C: Interaction Views, section Cryptographic Management).

Note that protection of file access should be thought of both in terms of read (access should only be given to the appropriate applications) and write (a component should not be able to use more storage than is allocated to it).

D.4.3.6 Human-Machine Interface

A PYRAMID deployment should, as required, provide a mechanism for generating information such as visuals and audio for Human-Machine Interface (HMI) output devices (e.g. rendering a graphical user interface to a display), and for obtaining user interaction events from HMI input devices (e.g. keyboards, mice, hands on throttle and stick (HOTAS), touchscreens, or microphones for speech input).

It is expected that as a minimum any graphics support should be through a primitive interface. Higher level functionality can be achieved using graphics libraries and toolkits.

For more information see Appendix A: PYRAMID Concepts, section Human-Machine Interface, and Appendix C: Interaction Views, section Operator Interaction Views.

D.4.3.7 Health and Usage Monitoring

System infrastructure faults (e.g. power failure) potentially impact the execution of every application on the Exploiting Platform. For more details see the Cyber Defence and Health Management PYRAMID concepts (Appendix A: PYRAMID Concepts, sections Cyber Defence and Health Management).

The infrastructure should provide a mechanism for reporting faults or anomalies identified in specific hardware and software. Those faults or anomalies that may impact system level function should be reported to health management components in the application layer (e.g. Anomaly Detection or Health Assessment) for assessment of this impact. Faults or anomalies which impact only the computing infrastructure can be analysed and resolved solely within the infrastructure. For an example as to how error reporting may be implemented, please see the Middleware Error IV (Appendix C: Interaction Views, section Middleware Error). For more information about investigation and anomaly detection, please see the Health Management PYRAMID concept (Appendix A: PYRAMID Concepts, section Health Management).

D.4.3.8 Software Configuration

Where software configurations concern the data that is related to the subject matter of the relevant component, this should be treated as data driving within that relevant component. The Data Driving PYRAMID concept (Appendix A: PYRAMID Concepts, section Data Driving) should be referred to in this case.

Where software configurations are related to software behaviour, e.g. buffer sizes, this information should be included at the infrastructure level. Configuration will need to be applied to each component, but common approaches may be determined at the Platform Specific System Development level. It may also need to be considered in subsequent processes for achieving certification or security accreditation relating to the software.

D.4.4 Certification and Accreditation

Whilst safety and security assurance objectives are different, the organisational and procedural issues are very similar. Safety and security assurance are properties that only exist for an entire product. An Exploiting Programme will have the responsibility for the analysis and verification activities, and completing the certification and security accreditation. For this reason, Exploiters should follow their normal safety and security processes. This section provides guidance in support of this and highlights where the PYRAMID Technical Standard, Ref. [1], and PYRAMID Technical Standard Guidance document contains safety and security related information for components.

The platform specific design needs to provide isolation of components and information as needed by the safety and security requirements of the deployment. The deployment approach needs to be flexible and support a number of strategies to allow these requirements for isolation of information to be satisfied. This flexibility can be enabled by segregating components so that certain data and functionality are partitioned, and then producing the data that enables the infrastructure to instantiate the required segregation. Segregation may be done for any reason, and by any means.

An Exploiting Programme should segregate components based on the isolation requirements identified as part of the platform specific design, and in line with the following recommendations:

- A component could be deployed in multiple variants or instances that could each be segregated from one another.
- If segregation is used, a component instance or variant should be kept wholly within its partition.
- For this purpose, an extension is treated as a component in its own right, subject to the rules defined in the Component Extensions PYRAMID concept (Appendix A: PYRAMID Concepts, section Component Extensions). PYRAMID does not preclude an extension being segregated from its parent or other extensions to the component.

If a deployment has no safety or security isolation requirements then the mapping of components onto hardware items within the infrastructure can be performed by considering the other needs of the deployment, such as resource allocation. A further complication that needs to be taken into account is that some other non-functional requirements might conflict with isolation requirements (for example requirements related to performance).

The need for partitioning may lead to updates to the deployment build set, for example to include multiple variants or instances of a component. This will require an update to the Platform Independent System Design (see the Multiple Instances of Components section for more details). This is illustrated in Figure 233: Example of partitioning for security purposes.

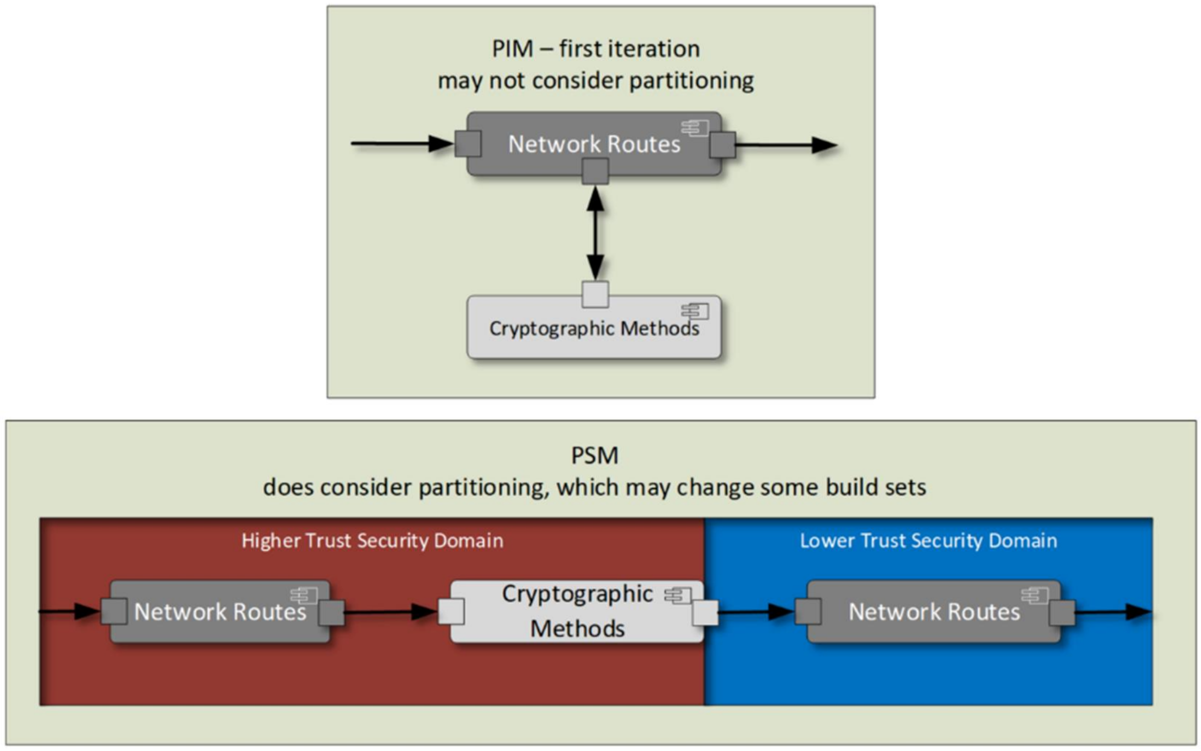

PIM – first iteration
may not consider partitioning
Network Routes
Cryptographic Methods
PSM
does consider partitioning, which may change some build sets
Higher Trust Security Domain
Lower Trust Security Domain
Network Routes
Cryptographic Methods
Network Routes

Figure 233: Example of partitioning for security purposes

Both security and safety requirements may drive certain components to be high assurance, and determining this in the Platform Specific System Development allows the processes used for developing these components to be tailored appropriately. DO-326A, Ref. [15], covers data requirements and compliance objectives, related to aircraft safety, for aircraft development and certification. Ref. [26] and Ref. [27] tackle the security assessment process and safety process.

D.4.4.1 Safety Considerations

The PRA has been designed with safety certification in mind. Each PRA component has been assigned an indicative Item Development Assurance Level (IDAL), which is the highest IDAL that the component is expected to have to achieve for a large (>5700kg) air vehicle (see Appendix A: PYRAMID Concepts, section Safety Analysis). The IDAL and associated analysis are provided for design assistance only, and do not place a requirement on the Exploiting Programme.

D.4.4.2 Security Assurance Considerations

Security controls should be identified as a result of a security risk assessment. Technical security controls are either security enforcing functions (SEF) or security related functions (SRF), and these are identified through a low level assessment and classification of data flows between components. A security considerations section is included within every component definition in the PRA, which helps identify whether the component is expected to participate in security related functionality. The security levels and associated functions are provided for design assistance only and do not place a requirement on the Exploiting Programme.

D.4.5 Data Management Design

The Platform Specific System Design should define system-wide approaches to the management of data within the deployment. This may include mechanisms to use data to configure components and bridges, through data-driving, and approaches to data synchronisation between multiple instances or variants of components.

D.4.5.1 Reconfiguration

A deployment needs to consider a mechanism for reconfiguring components, at run-time, to one of a predetermined set of configurations. This reconfiguration should be as transparent (or invisible) as possible to components which are not affected by the reconfiguration - it should not be something that they need to be aware of and any interactions they have with the reconfigured component should not be affected. For more guidance see the Data Driving PYRAMID concept (Appendix A: PYRAMID Concepts, section Data Driving).

Reconfiguration at run-time needs to be very carefully considered so that the reconfiguration does not result in safety or security issues arising as a component transitions from one configuration to another, or for the transition to be erroneously detected as an anomaly due to an error.

D.4.5.2 Data Synchronisation

A data synchronisation strategy enables multiple instances or variants of a component to be deployed and segregated whilst acknowledging that an amount of data needs to be synchronised between them.

Reasons data synchronisation is required include:

- Component instances or variants are separated by mission phase (e.g. mission planning and mission execution) in different locations, or to enable capability on multiple nodes within the Exploiting Platform (e.g. across multiple air vehicles).
- Load sharing, e.g. to stop bottlenecks of resource usage at a particular component instance and/or to optimise collective performance.
- Multiple instances or variants of a single component existing, e.g. for security partitioning, or physical segregation for safety redundancy (see the Certification and Accreditation section).
- Components that exist explicitly for the purposes of data replication, e.g. where parts of a system are aligned for backup or redundancy. Communications components could be used for this purpose; see the Use of Communications PYRAMID concept (Appendix A: PYRAMID Concepts, section Use of Communications) for more information on how these components can be used.

Services need to be defined to support data synchronisation. These should follow pre-defined patterns, e.g. ‘peer-to-peer’ synchronisation or synchronisation via a common data source (e.g. database). The choice of pattern should take account the ways in which the components will be deployed (for example whether they will be distributed across multiple nodes) in order to take account of relevant system constraints, such as bandwidth and latency.

D.4.5.3 Inter-Application Communications

In all but the most simple cases, components will be distributed across multiple computing resources within the infrastructure architecture. Therefore communication mechanisms to allow communication between these different computing resources need to be provided. Each component's provided and consumed services are connected by bridges, which are able to access the underlying communications mechanism provided by the software infrastructure, allowing the components to communicate.

The bridges could be data-driven to allow configuration of the communications mechanism used to communicate between components. This approach would enable flexibility to support differing phases of the deployment development, for example to support system and integration testing by allowing the bridges to be reconfigured to support a particular test configuration of the system that might be needed before all components become available. A data-driven configuration approach may need to be supported by further data management to help manage change across large numbers of variables. For example, an offline dynamic modelling tool for identifying interactions between variables.

Note that when assessing the integrity required to achieve the desired safety or information assurance goals, the level of assurance required for the bridges between components will also need to be assessed.

D.4.6 Approach to Component Services

Component services will be defined as part of the component specifications in the Platform Independent System Design (see the Services section). The mode of inter-component communications is a fundamental characteristic of interactions that use these services, and the two modes that are typically chosen between are synchronous (request response) or asynchronous (event driven). Choosing one mode to use as an approach for service development for the entire deployment (or all components being specifically developed for that deployment) will simplify bridges and assist with system integration. The choice of communications mode is likely to be based on the environment that the system is to be used in. See Ref. [9] for more details.

D.4.7 Architectural Alignment and Interoperability

Architectural alignment is the process of aligning a deployment's architecture (e.g. the breakdown of the deployment into components and the interfaces between them) to the architecture of other (external) systems.

This alignment supports the sharing of best practice and the pursuit of interoperability through the use of patterns and consideration of component responsibilities and data exchanges. For an example of how to support interoperability with other systems, see the Tactical Exchange interaction view (Appendix C: Interaction Views, section Tactical Exchange).

The Platform Specific System Design stage may include the determination of the approach to architectural alignment, although if the build set needs to be adjusted to align with this approach, then this would need to be covered by Platform Independent System Design.

When considering other architectures, it is important to be cognisant of the roles and responsibilities of PYRAMID components in order to ensure that alignment with external components doesn’t conflict with PRA component definitions. Although there may be little or no control over external components, the internal alignment should not blur component boundaries or introduce tight coupling.

D.4.8 Infrastructure Implementation

As part of the PSM the technology dependent detail needs to be captured and included in the design; this includes identification of low level communications protocols and underlying operating systems and libraries. The design abstraction should not be confused with ambiguity; from a system point of view the aim of the modelling of the Platform Specific Implementation is to generate an unambiguous definition of what the component interfaces are, and what the component requires from the infrastructure. By working with Component Developers the service definitions can be aligned, reducing the bridging effort.

Safety and security targets still need to be met in the infrastructure that supports the components; when implementing a bridge, the DAL of the components whose service interactions the bridge supports need to be considered. Operating systems and middleware also need to support the DAL requirements of their hosted components.

D.4.9 System Reuse

There is always more than one route to a final product and a new system is never designed in a vacuum. There may be existing systems, such as legacy systems, that can be fully or partially reused within the Exploiting Platform being developed. Such existing systems may range from specific equipment, potentially including software, to significant portions of a wider system, such as an air vehicle subsystem.

The reuse may range from reusing blocks of code to reusing hardware and software that can be treated, through its own interface requirements, as an independent subsystem.

Although these considerations are not unique to Platform Specific System Development, and potential reuse should be identified when considering platform independent design, there are specific considerations that need to be taken into account during Platform Specific System Development that may require a different approach to other aspects of the Exploiting Platform. For example, a specific communications protocol may be needed to interact with a reused subsystem.

PYRAMID components are deliberately designed to encompass isolated areas of functionality to increase the potential for reuse, making it likely for legacy code to encompass the functionality provided by a set of components. Although it is the intention that artefacts can be reused at any stage of the design process, reuse of PSM artefacts is typically more difficult due to the greater detail of platform specific specialisation and dependencies by that stage. Therefore if it is an aim to generate feasibly reusable PSM artefacts, both those working on the PIM and PSM should consider how their decisions may affect this reusability. Most decisions about the functionality of reused artefacts are appropriate earlier, at the PIM level, before these decisions are implemented at the PSM level.

It is important to note that reuse of developed components may adversely affect the security of Exploiting Platforms. The use of a developed component (or fragments thereof) on both secret and lesser classified platforms has the potential to compromise the secret one, therefore care must be taken in this respect.

Where a deployment aims to reuse PYRAMID compliant components, due consideration must be given to the potential impact of changes to PRA component definitions between different versions of the PRA (see section, Different Versions of the PRA).

D.4.10 Model Transformation

The considerations outlined in the Platform Specific System Development section can be viewed as a generic set of guidelines and recommendations applicable across all components that map the PIM to the PSM. As these guidelines are disconnected from the middleware or operating system solution and final messaging protocols, they can be captured in a transformation specification, which itself forms the requirements for a model translator. The model translator can then be used in a transformation engine to do the mapping and automatically generate the PSM, as summarised in Figure 234: PIM to PSM Mapping, which has been adapted from Ref. [28].

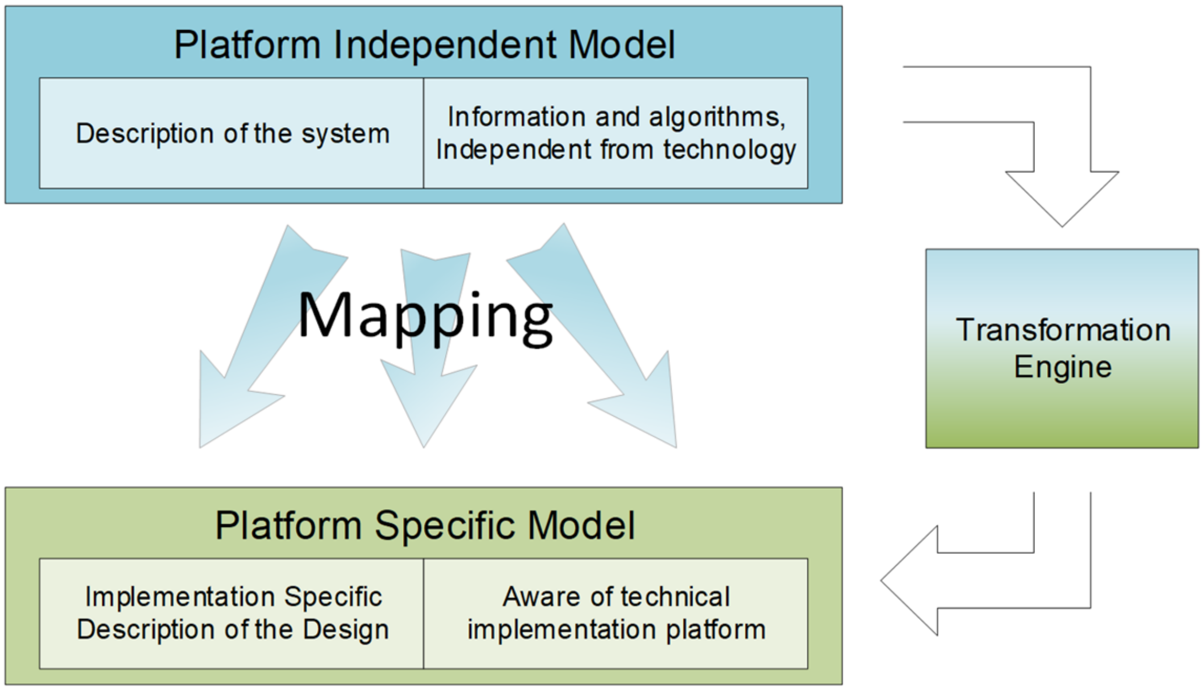

Platform Independent Model
Description of the system
Information and algorithms, Independent from technology
Mapping
Transformation Engine
Platform Specific Model
Implementation Specific Description of the Design
Aware of technical implementation platform

Figure 234: PIM to PSM Mapping

Automatic transformation of PIM to PSM ensures a consistent approach for all components, and allows the impact of changes to the PIM to be captured. A PSM can still be inspected, allowing any inspection of the detailed design before it is committed to code. This is particularly important in high assurance level implementations.

Different transformation specifications can be used to generate different PSMs for different target systems, allowing a family of PSMs to be generated for different uses. Automatic transformation can be taken a step further with the PSM being used to automatically generate code for the structure of component implementations and the integration of components with the Exploiting Platform infrastructure.

The adoption of standards-based shared meta-models across the deployment is more likely to allow for the sharing of information between design tools supported by different vendors. If this approach is adopted across multiple Exploiting Programmes then they can be quoted along with the PYRAMID model to define a de facto standard. For example, by using the FACE® Technical Standard, Ref. [20], it is possible to:

- Map conceptual data elements to the deployment-specific elements, automatically identifying logical and physical representations on all sides of an interface linked by a common conceptual meaning.
- Select suitable logical representations or translations that reflect physical phenomena from a library of predefined logical data types. This can support future reuse.
- Select a subset of logical representations from the existing library, which will reduce the number of logical conversions needed, and will support the maintenance of bridges within the deployment.
- Generate interface control documentation that matches the modelled interface definition. This is particularly powerful where the final interface is also auto-generated from the model. If all documentation is automatically generated from the model, this should help avoid the staleness that often comes during later revisions of the product where the documentation is not updated.

D.4.11 Good Practice

This section provides good practice recommendations applicable to Platform Specific System Development, including guidance for avoiding common pitfalls.

Although the recommendations in this section are most relevant to Platform Specific System Development, they may also be relevant to other stages of the design and development of a deployment.

D.4.11.1 Bridge Through Middleware

What is treated at the platform independent level as a simple peer-to-peer interaction between components may occur between physically segregated areas, as illustrated in Figure 235: PIM Bridging and PSM Bridging between Segregated Areas. Part of the job of bridges is to hide the complexity of communications from components. A component should be unaware of the source of information, although the service implementation needs to be designed to cope with latencies or delays resulting from sourcing information from a distance away.

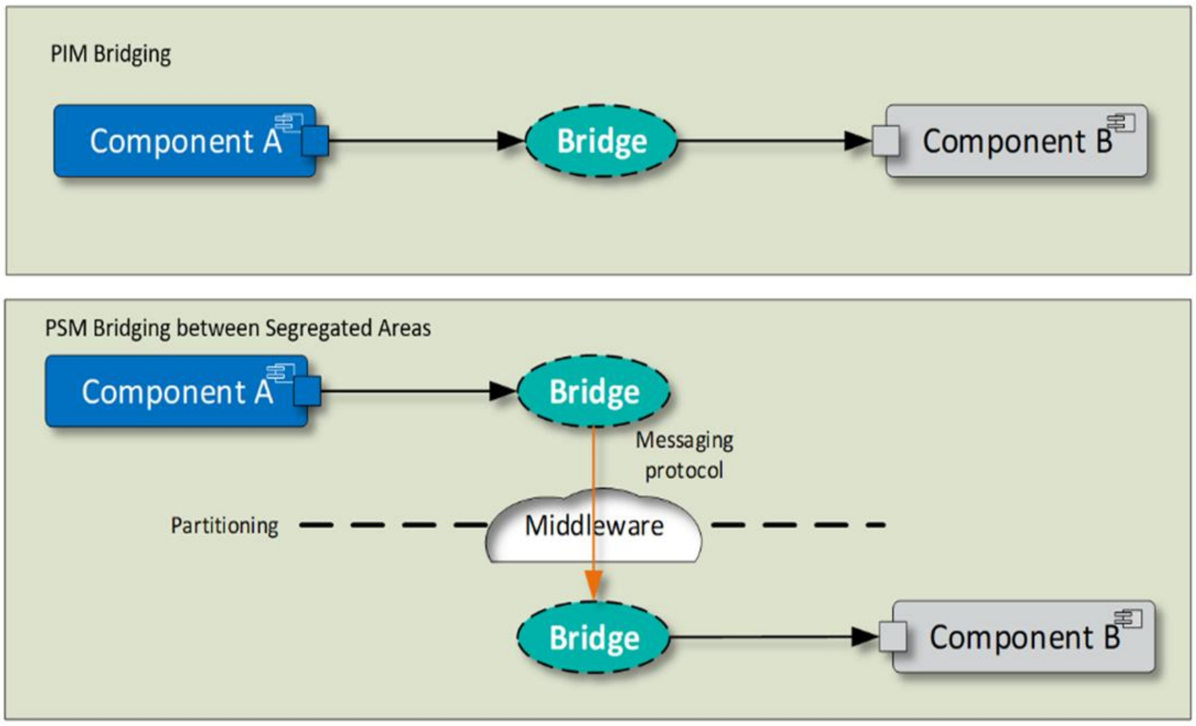


Figure 235: PIM Bridging and PSM Bridging between Segregated Areas

The concept of a service being separated from the internal working of the component is to allow the component to carry on activities while the related service deals with the bridge and any data exchange. If components cannot be decoupled in a particular platform (i.e. a component will not be able to carry on other activities until it has received a response from another component), consideration should be given to co-locating them when deploying them.

Additionally, where PSM bridging is used between segregated areas particular consideration should be given to the effect on the component's operation if the interconnection is lost.

A common middleware interface / messaging protocol should be used to aid with cross platform re-usability; see the Use Other Standards and Models in Conjunction with the PRA section for more on the use of additional standards and reference models.

D.4.11.2 Separate the Concept of 'What Data Is' from How it is Acquired

The PIM assumes that there are no interface delays - all exchanges are instantaneous and all data that is in the domain of the component exists in the component. These things do however need to be considered as part of the platform specific modelling, including where data is created. It is the responsibility of all the components to manage their own data, including data loading (see Appendix A: PYRAMID Concepts, sections Operational Support and Storage). However, a common approach to data loading should be identified and implemented at the PSM level, as many aspects of it will be infrastructure dependent.

D.4.11.3 Coordinate Preparation and Execution Interactions

As part of the Platform Specific System Design, concerns around how to manage resources and latencies need to be taken into account when considering component interactions. There is a particular issue in how allocated resources are matched to their intended usage at execution.

The solution is likely to involve a planned action that is dependent on its associated resource needs, each of which are associated with an associated activity of a resource.

The association cannot be determined implicitly in bridges as this is a problem concerning recall of earlier preparation (in this instance, allocation) information. If the identifiers were contained within the bridge, the user of the resource would still have to identify something to the bridge that would enable it to recall the relevant planned action. This means the System Designers need to consider how identifiers are managed as part of component design and capture this, otherwise, different parts of the system will not work together.

D.4.11.4 Capture and Share Platform Specific Information

It is important that platform specific information is captured and shared early in the process. This enables component design and development to consider platform specific aspects when needed, and is especially important for aspects common across components. Artefacts such as standard documents or shared models may be used to capture this information into a single record source, which can then be referred to when needed in component specifications.

D.4.12 Summary

An iterative approach for Platform Specific System Design is expected to be followed that will involve looking at the execution environment and its impact on the Platform Independent System Design, as well as the integration of components as they are produced (see the Platform Specific Component Development section).

Artefacts produced during this stage should be appropriately captured (e.g. in platform specific equivalent versions of a component specification) and shared with Component Designers and Component Developers. The mechanisms for this should enable close regular interfacing between any separate parties. Some of the typical artefacts produced or considered during this stage are:

- **Non-functional requirements**, which need to be considered to ensure they are met by the system; see the Non-Functional Requirements section.
- **Infrastructure definition**, which needs to be considered as it specifies the infrastructure interfaces, middleware and operating system chosen to support the components; see the Infrastructure Architecture section.
- **A partitioning approach**, produced to define how components should be segregated or aggregated to meet safety, security and performance requirements; see the Certification and Accreditation section.
- **An approach to configuration**, and other management of data (produced); see the Data Management Design section.

- **A component services approach**, produced to define the underlying protocols and mechanisms that have been chosen to tie services together; see the Approach to Component Services section.

D.5 Platform Specific Component Development

D.5.1 Introduction to Platform Specific Component Development

The Platform Independent Component Design stage defined the required component functionality without consideration for how it was to be achieved (see the Platform Independent Component Design section). The next step of the component development approach is Platform Specific Component Development, which involves the consideration of the non-functional requirements and the infrastructure specification at the level of the individual components.

One option for the Platform Specific Component Development stage is to evolve the PIM component specification to produce a platform specific component specification as part of the PSM. Additionally, a suitable approach for verification and integration of platform specific components should be agreed. The contribution towards safety, security and other non-functional requirements provided by a component or the infrastructure should also be discussed and updated as development continues.

D.5.2 Infrastructure Considerations

D.5.2.1 Extensible Data

The Data Driving PYRAMID concept (Appendix A: PYRAMID Concepts, section Data Driving) describes how data can be specified for operational use (e.g. to provide the weapon fit for a mission). Data can be delimited or otherwise encoded in a way which enables extension of the data without having a direct impact on components that are making use of the data.

Extensibility can be achieved by relating individual parts of the data with an identifier or tag along with an indication of the extent of the data. This allows extension components, for example, to identify which parts of the data they need and ignore data that they do not need. Data can therefore be extended to support a particular component or extension component requirement without affecting existing interface integrations.

An example is the use of Session Description Protocol (SDP) subunits within the Session Initiation Protocol (SIP), Ref. [29], to define supported capability. If the receiver supports the capability it can decipher the message; if it does not it can understand enough to say that capability is not supported.

When selecting a format for the data, care should be taken to consider whether the adoption of data representation standards may result in the need for safety certifiable data translation library code. Safety certifiable library code with appropriate verification artefacts may need to be supplied to all Component Developers to avoid duplication of effort on an Exploiting Programme. Similarly, the security aspects of processing data need to be given appropriate attention.

D.5.2.2 Health Monitoring and Fault Management

During the Platform Independent Component Development phase (described in the Platform Independent Component Design section) it is assumed that the component will execute the expected behaviour, and all the supporting infrastructure can be fully relied on. In the platform specific phase enough is known about the infrastructure (e.g. the operating system, application software infrastructure, or hardware) to make judgments on what can and cannot be relied on, and what the development implications are.

D.5.3 Component Implementation

Given a set of requirements, there are many ways in which to realise a component and these are not covered at any length in this appendix. However, it is recommended that any code developed with a PYRAMID deployment in mind should be data-driven (provided that it is done in a way that is feasible within the development time and cost constraints). This means that representative data sets should be provided to any System Integrator.

D.5.3.1 Legacy Code Reuse

The PRA is primarily expected to enable reuse of components, whether that be implemented PYRAMID components or component-level design artefacts, with the goal of saving time and effort in a development programme. However this does not rule out reuse of code that was not originally developed as part of a PYRAMID component within a PYRAMID deployment. Below is a summary of considerations for code reuse based on the guidance in the FACE® Reference Implementation Guide, Ref. [50], which covers similar issues:

- Determine the grouping and applicability of components for the legacy system:
  - Split the legacy system into multiple components or keep it as a single deployment?
  - Is the software still supportable?
  - Can the software be adapted to the PRA?
- Characterise and assess the legacy software functionality (the FACE® Technical Standard, Ref. [20], describes this functionality as software-based "capability"):
  - Was the functionality designed for capability monitoring, safety, or security?
  - What other functionalities does this code interface to?
  - What types of interfaces are there and what standards were used for those interfaces?
- Assess how the functionality fits into the PRA:
  - Have component group boundaries been defined for the functionality?
  - To what protection domain(s) do the component group(s) belong?
- Determine how to integrate the functionality into the architecture without polluting components:
  - Should the legacy architecture be converted to align to the PRA?
  - Should the legacy architecture interface to new PYRAMID deployment functionality hosted within the existing architecture?
  - Should the legacy architecture interface to new PYRAMID deployment based system functionality hosted within an architecture aligned to the PRA?

Where refactoring of legacy code would be required, the expected costs should first be evaluated against the benefits of refactoring the code. Reverification, capturing of safety artefacts (see the Manage Safety Evidence Development section), and capability update issues should be considered where applicable. Additionally, where legacy code from a standard product is being specialised or tailored during integration, the full impact of this should be carefully considered to weigh the benefits and issues, such as future support challenges arising from changes to the product.

D.5.3.2 Automatic Code Generation

The traditional approach to software development (see Figure 236: Traditional Software Development Lifecycle) was to treat design and development as elaborative process steps, where “fidelity” or levels of detail were added during the development stage that were considered ‘too low level’ to be included in the analysis or design stages. This often led to the models being used to capture the analysis of the problem or the design of the system being vague or incomplete (Ref. [6]).

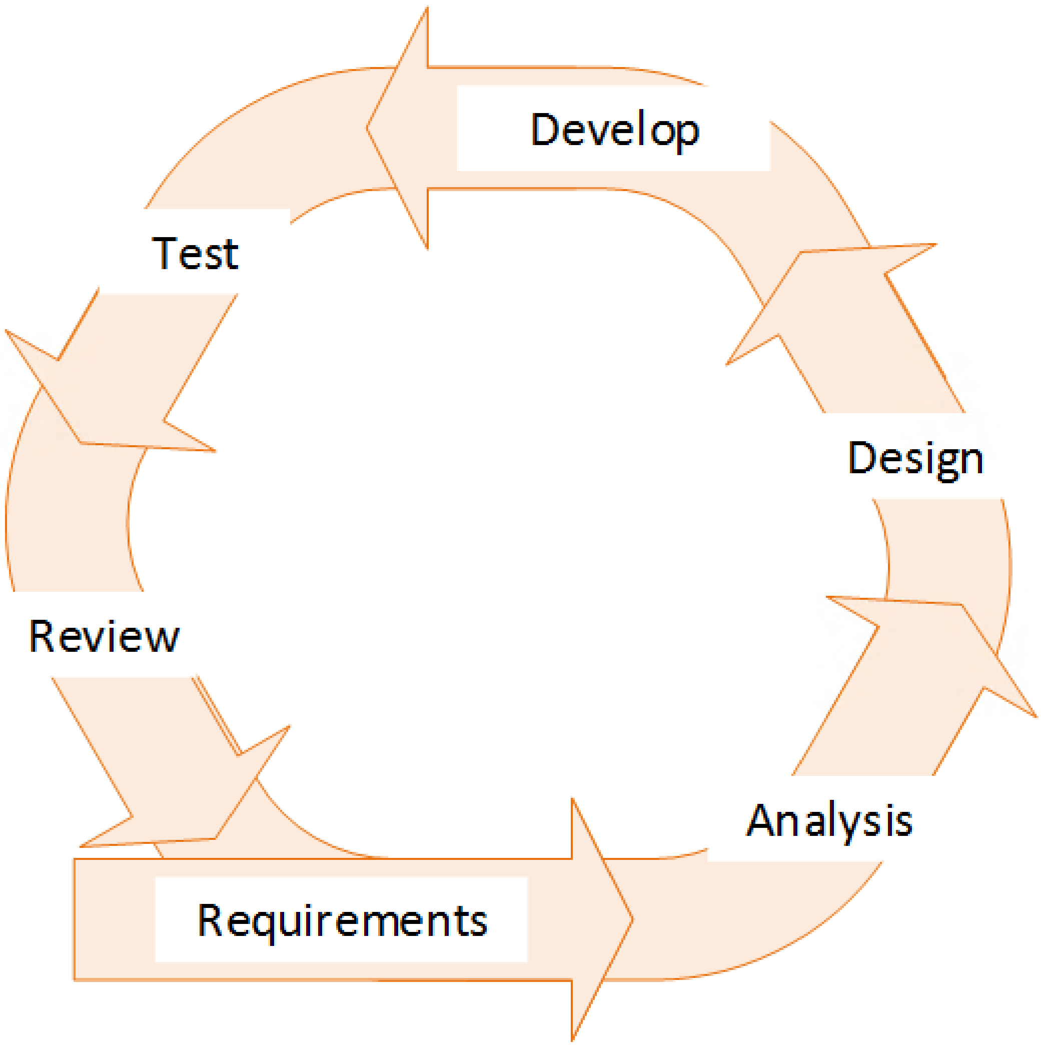


Figure 236: Traditional Software Development Lifecycle

By moving some of the rigour into the model used to capture system design it is possible to automate the generation of large parts of the code. This improves the consistency of code, facilitates documentation of behaviour (in the model), and, dependent on toolchain support, allows for faster rework. Transformation engines can be used to generate Platform Specific Models from Platform Independent Models (see the Model Transformation section), and automatic generation of code can then take place from the Platform Specific Model to allow code to be produced that is targeted at the specific execution platform.

When using auto-generation of code from a model, it is likely that any validation rules required for the resultant system still need to be applied, e.g. DO-326A, Ref. [15]. For details on how to use a Model Driven Architecture (PYRAMID deployment) approach to support a model based approach to software development see Ref. [6].

Note that executable code is not the only thing that can be generated from the model. Wherever possible, documentation should also be generated from the model, allowing the model to be the single source of truth. Data models and interfaces can be used instead of, or to produce, interface control documents.

D.5.4 Working with System Integrators

Modelled components are, by design, not complete applications in their own right. Therefore individual components, or groups of components, will always require integrating into a larger deployment. A decision has to be made through discussion with the System Integrator on component groups to be delivered as ‘mini-deployments’, and whether internally used services and interfaces are defined at a PIM or PSM level (even if not accessible in the executable software) to allow reuse of the modelled components in a different configuration.

To give examples of context for a component of interest, in order to determine potential broader use of the component in other deployments, Component Developers can use PYRAMID interaction views (IVs) (see Appendix C: Interaction Views). System Integrators of other potential deployments should be engaged at the earliest opportunity.

D.5.5 Good Practice

This section provides good practice recommendations applicable to Platform Specific Component Development, including guidance for avoiding common pitfalls.

Although the recommendations in this section are most relevant to Platform Specific Component Development, they may also be relevant to other stages of the design and development of a deployment.

D.5.5.1 Consider Appropriate Middleware Access

While a component will access the resources provided by or via middleware, this interaction must never be:

- Beyond the scope of the component’s subject matter;
- Instead of using the component’s defined interface.

A component can directly interact with middleware where it is within the scope of that component to do so, for instance directly accessing stored data when it is owned by the component. This is true no matter what role a component has within the control architecture.

D.5.5.2 Manage Safety Evidence Development

An agreement between Component Developers and System Designers on how safety evidence will be developed and managed should be reached early in the development process, so that dependencies between the different stages of development of the deployment can be identified. The agreed management methods should cover the mechanisms for provision of any applicable safety evidence from a Component Developer to the System Designer. Frequent information sharing between the parties is good practice, and is especially important regarding safety related developments.

D.5.6 Summary

This section covered additional considerations that help mature a component based on execution platform considerations. The expectation is that this will be an iterative process, and the development of the component will be done in parallel to the system design and integration discussed in the Platform Specific System Development section, the final goal being to generate executable software. The most important point discussed in this section is **Component Implementation**, either through software development, code reuse or automatic code generation; see the Component Implementation section.

D.6 Different Versions of the PRA

One of the primary goals of PYRAMID is to maximise the opportunities for the reuse of components within and across Exploiting Programmes. Whilst it is assumed that an Exploiting Platform will be contracted against a specific PRA version, components that are identified as candidates for reuse, ‘off-the-shelf’, may not have been developed to the required PRA version or compliance read across may not be provided.

The PRA has been developed over many years and has reached a level of maturity and stability based on considerable analysis, testing, validation and the incorporation of industry feedback. However, where the need arises, a new version of the PRA could result in changes to PRA component definitions that impact the compliance of existing PYRAMID components (if chosen to be assessed against the later PRA version). Changes to the PRA may modify the scope of the responsibilities of individual PRA components, or add or remove PRA components. Furthermore, some Exploiters may have developed components against early, less mature, versions of the PRA, prior to it becoming a standard.

Proposals for the reuse of existing PYRAMID components will therefore have to consider the implications of any such updates to the PRA. In addition, where a new version of the PRA is issued during the lifecycle of a project, the costs and benefits of transitioning to the new version of the PRA will need to be assessed.

A deployment containing PYRAMID components developed against different versions of the PRA may experience incompatibility between PYRAMID components due to scope changes in the associated PRA components' responsibilities. In these circumstances, a deployment may not be able to fully realise the benefits of the PYRAMID approach, as there may no longer be a correct and consistent separation of subject matter concerns between PYRAMID components.

The following examples discuss the potential impact of revisions to the PRA and how this may be managed.

D.6.1 Change to Responsibilities of PRA Components

This example considers a simple change to the scope of two PRA components, with some of the responsibilities of one PRA component becoming the responsibility of a second PRA component, thus changing the subject matter demarcation between the two.

Figure 237: Change to Responsibilities of PRA Components shows this example, in which the PRA is updated from version 1 to version 2. In version 2 the scope of PRA component B has increased to incorporate some of the responsibilities of PRA component A. The scope of PRA component A is therefore correspondingly reduced. The example assumes that two PYRAMID components, A and B, have been developed and are compliant with version 1, where each has implemented a subset of the full scope of responsibilities defined for the corresponding PRA component. As a result of the revisions to the PRA under version 2:

- PYRAMID component A is directly impacted by the change and is not compliant with PRA version 2 since it incorporates functionality that now falls within the scope of the responsibilities of PRA component B. In order to make PYRAMID component A compliant with version 2, this functionality would need to be removed (not shown) and the component requalified.

- PYRAMID component B is not directly impacted as a result of the scope of responsibilities of PRA component B being increased. PYRAMID component B therefore remains compliant under PRA version 2. However, assuming that a deployment utilising PYRAMID components A and B requires the full scope of the functionality originally implemented under version 1, PYRAMID component B would need to be expanded to incorporate the functionality removed from PYRAMID component A (not shown).

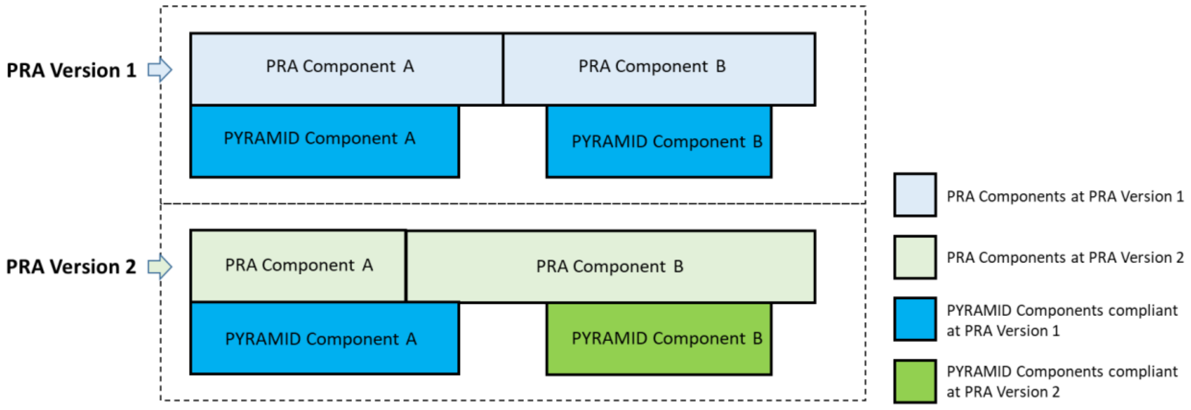


Figure 237: Change to Responsibilities of PRA Components

The reuse of PYRAMID components may be undermined where these are not maintained in line with the latest version of the PRA. Depending on which PYRAMID components are developed at version 1 and version 2, the following issues may occur if consistent PRA versions are not used:

- If only PYRAMID component A is developed at version 1, and not subsequently upgraded to version 2, this may result in duplicated functionality if used in the same deployment as PYRAMID component B developed to version 2. This is depicted in Figure 238: Potential Issues with Change of Subject Matter Scope '(i) Duplication', and may result in an increase in system complexity with corresponding maintenance overheads, or redundant code in one component with the associated implications for qualification. Additionally, this may also lead to incompatibility or errors across the system, as the same capability may be performed differently in different parts of the system, e.g. if older versions of PYRAMID component A continue to use their own capability and not utilise the capabilities of the new PYRAMID component B.
- If only PYRAMID component B is developed at version 1 and not subsequently upgraded to version 2, this may result in a potential functional gap when taken in conjunction with PYRAMID component A developed to version 2. This is depicted in Figure 238: Potential Issues with Change of Subject Matter Scope '(ii) Potential Gap', and would necessarily have to be resolved by expanding the functionality of PYRAMID component B in accordance with PRA version 2.
- If both PYRAMID components A and B are developed at version 1, but only one is subsequently upgraded the same situations can occur.

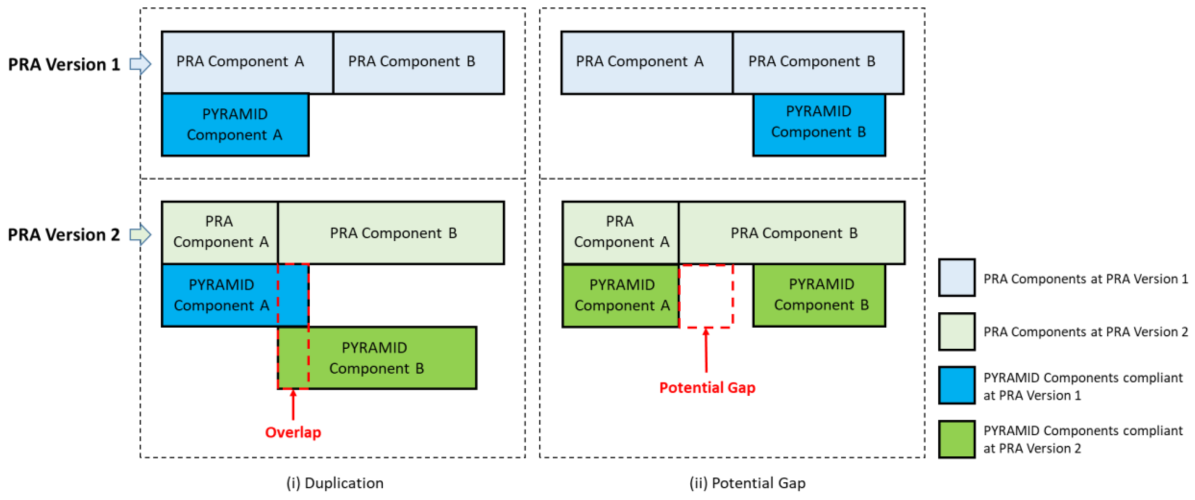

PRA Version 1
PRA Version 2
PRA Component A
PRA Component B
PYRAMID Component A
PYRAMID Component B
Overlap
Potential Gap
(i) Duplication
(ii) Potential Gap
PRA Components at PRA Version 1
PRA Components at PRA Version 2
PYRAMID Components compliant at PRA Version 1
PYRAMID Components compliant at PRA Version 2

Figure 238: Potential Issues with Change of Subject Matter Scope

D.6.2 Addition of New PRA Components

This example considers the addition of a new PRA component to the PRA. This could occur if the scope of the PRA is expanded, or if a common area of functionality is identified that warrants a new distinct subject matter.

Figure 239: Addition of New PRA Components shows this example, where a new PRA component C, is created from a common area of functionality separated out from within the scope of two existing PRA components, A and B (with these two PRA components consequently being reduced in scope). The example assumes that two PYRAMID components, A and B, have been developed and are compliant with version 1, where each has implemented a subset of the full scope of responsibilities defined for the corresponding PRA component. As a result of the revisions to the PRA under version 2:

- PYRAMID component A is directly impacted by the change and is not compliant with PRA version 2 since it incorporates functionality that now falls within the scope of the responsibilities of PRA component C. In order to make PYRAMID component A compliant with version 2, this functionality would need to be removed and the component requalified.
- PYRAMID component B is not impacted since the functionality embodied in PYRAMID component B is not within the scope of responsibilities moved to PRA component C. PYRAMID component B therefore remains compliant under PRA version 2.
- A new PYRAMID component C would need to be developed, in accordance with the definition of new PRA component C, where a new or existing deployment requires this capability and aims to be compliant with PRA version 2.

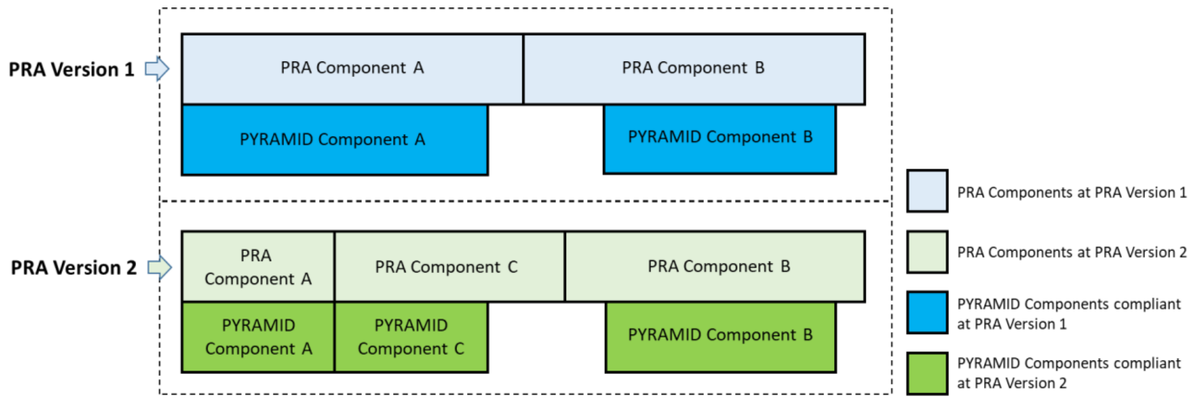


Figure 239: Addition of New PRA Components

As with the previous example, the reuse of PYRAMID components may be undermined where these are not maintained in line with the latest version of the PRA. Depending on which PYRAMID components are developed at version 1 and version 2, the following issues may occur if consistent PRA versions are not used:

- If only PYRAMID component A is developed at version 1, and not subsequently upgraded to version 2, this may result in duplicated functionality if used in the same deployment as the new PYRAMID component C. This may result in an increase in system complexity with corresponding maintenance overheads, or redundant code in one PYRAMID component with the associated implications for qualification. Additionally, this may also lead to incompatibility or errors across the system, as the same capability may be performed differently in different parts of the system, e.g. if older versions of PYRAMID component A continue to use their own capability and not utilise the capabilities of the new PYRAMID component C.
- If only PYRAMID component B is developed at version 1, this does not cause a problem since the functionality in B has not embodied functionality moved to PRA component C under version 2. Where a deployment requires the capability defined for PRA components A and C, the required PYRAMID components can be developed in accordance with PRA version 2 and used in conjunction with PYRAMID component B.

D.6.3 Removal of Existing PRA Components

This example considers the removal of a PRA component from the PRA. Just as PRA components could potentially be added between versions of the PRA, they could also be removed if they are determined to be no longer required.

Figure 240: Removal of Existing PRA Components shows this example, in which the PRA is updated from version 1 to version 2. In version 2, PRA component C has been determined to be no longer required and has been removed from the PRA and some of the responsibilities of PRA component C are incorporated into the responsibilities of PRA components A and B. The scope of PRA components A and B are therefore increased accordingly. The example assumes that three PYRAMID components, A, B and C, have been developed and are compliant with version 1, and that each has implemented some or all of the full scope of responsibilities defined for their corresponding PRA component. As a result of the revisions to the PRA under version 2:

- PYRAMID component A is not directly impacted since the scope of responsibilities of PRA component A have increased. PYRAMID component A therefore remains compliant under PRA version 2.
- PYRAMID component C can no longer be included in a deployment aiming to be compliant at PRA version 2 as there is no longer a corresponding PRA component to be compliant to. If the functionality previously provided by PYRAMID component C is required by such a deployment then an update to PYRAMID component A will be required (as shown).
- PYRAMID component B is not directly impacted since the scope of responsibilities of PRA component B have increased. PYRAMID component B therefore remains compliant under PRA version 2. However, if a deployment utilising PYRAMID components A and B required the full scope of the functionality now associated with PRA component B, PYRAMID component B would need to be modified to incorporate the functionality previously associated with PRA component C (not shown).

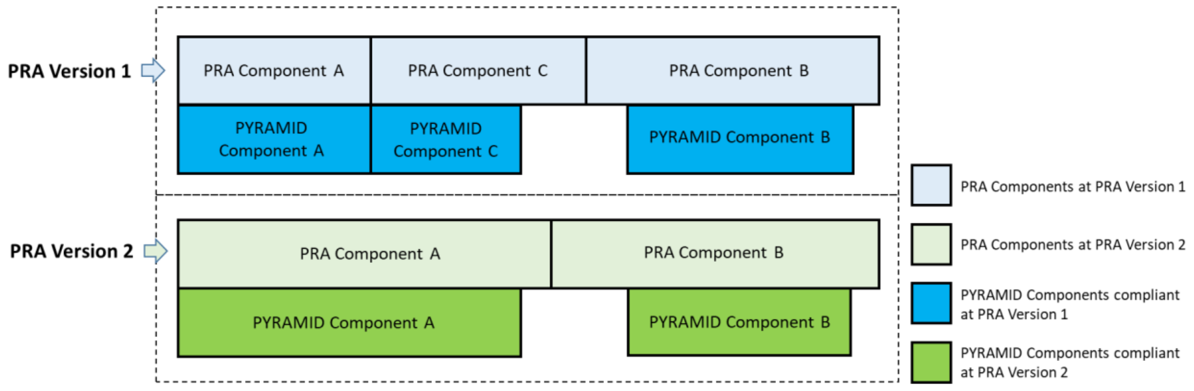


Figure 240: Removal of Existing PRA Components

As with the previous example, the reuse of PYRAMID components may be undermined where these are not maintained in line with the latest version of the PRA. Depending on which PYRAMID components are developed at version 1 and version 2, the issues identified in the earlier examples may occur if consistent PRA versions are not used.

D.6.4 PRA Version Description Document

A Version Description Document, which identifies the changes between baseline versions of the PRA, is provided with each version of the PRA. The Version Description Document is an exploiter's primary guide to understanding the changes between PRA versions, including which PRA components are impacted and the scope of the change.

Appendix E: Compliance Guide

E.1 Introduction

This appendix discusses the goal of PYRAMID compliance and provides supporting guidance material in relation to the compliance rules, including how compliance with the rules might be achieved, demonstrated and assessed.

The rules for achieving compliance with the PYRAMID Technical Standard are defined in PYRAMID Technical Standard, Ref. [1], section How to Comply with the PYRAMID Technical Standard. In order to establish that the PYRAMID approach has been adhered to, the standard defines compliance rules for individual components, component connections and deployments.

The following key terms are used in this appendix:

- **PRA Component:** A PYRAMID reference artefact, defined by a role, a distinct set of responsibilities, entities and services, for a specific, discrete area of subject matter.
- **PYRAMID Component:** A component that is intended to comply with a PRA component definition.
- **Target PRA Component:** A PRA component against which the compliance of a PYRAMID component is being determined.
- **PYRAMID Deployment Scope:** The elements of a deployment that are intended to comply with the PYRAMID Technical Standard.

E.2 PYRAMID Compliance Context

This section provides context for PYRAMID compliance, including its goal and how this fits into the development lifecycle of Exploiting Programmes.

E.2.1 Compliance Goal

The goal of PYRAMID compliance is to help ensure that an Exploiting Programme realises the benefits provided by the PRA. In particular, compliance seeks to preserve the separation of subject matter, a concept that has been adopted by PYRAMID to enhance the rapid development and upgradability of future platforms. PYRAMID component compliance is therefore based on the absence of subject matter pollution. Each PRA component represents a discrete area of subject matter. Subject matter pollution occurs when a PYRAMID component includes any subject matter of a PRA component that it is not based on. The components of a PYRAMID compliant deployment should have no subject matter pollution.

E.2.2 PYRAMID Deployment Scope

The scope of a deployment for which compliance will be assessed belongs to the Exploiting Programme. It is highly likely that a deployment is never going to be exclusively derived from PYRAMID and parts of the system capability that could have been provided in accordance with the PRA may be delivered by other means, e.g. using legacy software, middleware or dedicated equipment. Equally, PYRAMID derived system elements, need not always be implemented in the form of application software, e.g. FPGAs. Only the parts aligned to the PRA can be PYRAMID compliant. This section of the deployment is called the PYRAMID deployment scope. Figure 241: Example PYRAMID Deployment Scope shows the PYRAMID deployment scope section of an example deployment. As shown in the diagram, the PYRAMID deployment scope includes PYRAMID components and the component connections between these components. Exploiters should identify the PYRAMID deployment scope section of their deployment early on in the deployment lifecycle.

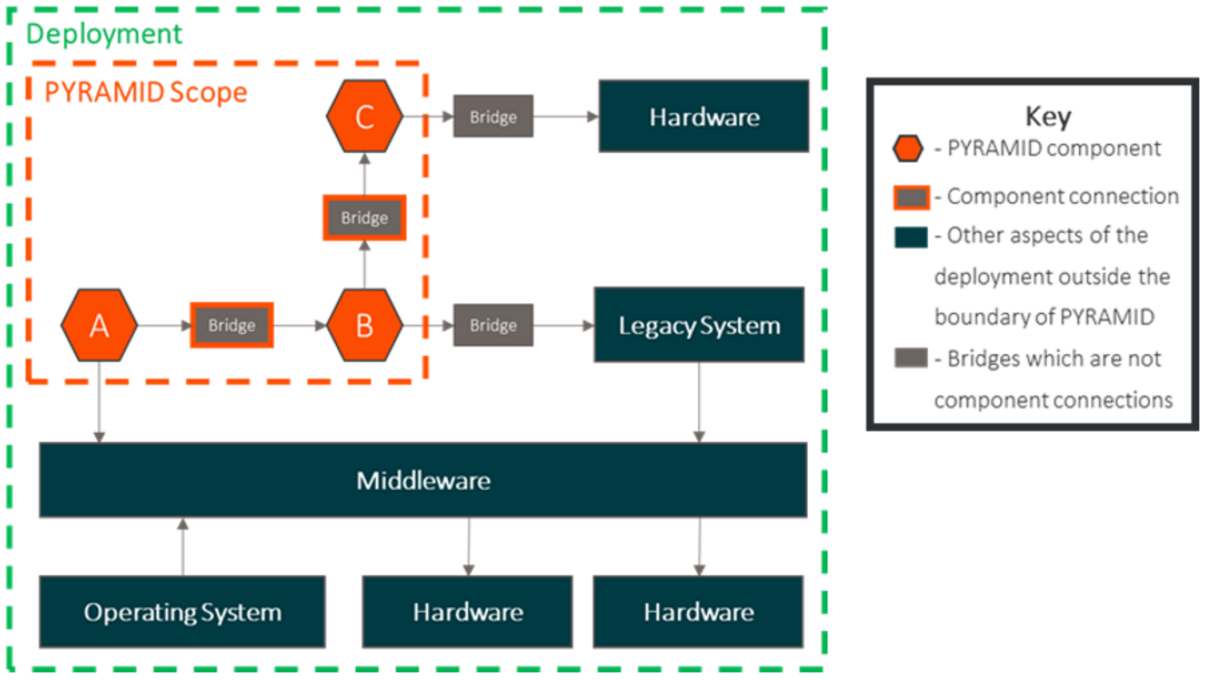


Figure 241: Example PYRAMID Deployment Scope

E.2.3 The Importance of Compliance Declaration

In order to generate an argument that a deployment, or the components within it, has been developed following the PYRAMID approach, it is essential for exploiters to be able to declare compliance against the PYRAMID Technical Standard. In providing compliance information (on a compliance declaration), an Exploiting Programme communicates in a consistent manner:

- An understanding of whether each component within the deployment is subject matter compliant in accordance with the PYRAMID Technical Standard.
- An understanding of whether the component connections within the deployment are compliant with the PYRAMID Technical Standard.
- An understanding of whether the deployment as a whole is compliant with the PYRAMID Technical Standard.

In addition, a compliance declaration should define the functional scope of the PYRAMID elements of an Exploiting Programme, by defining:

- The set of PYRAMID components and component connections developed for that Exploiting Programme.
- The functional scope of each PYRAMID component, including the aspects of the target PRA component that have been incorporated.

A compliance declaration should be against a specified issue of the PYRAMID Technical Standard.

It is expected that a compliance declaration will contain the following information as a minimum:

- Exploiting Programme name
- Developer name
- Deployment/Component/Connection name
- Deployment/Component/Connection version
- PYRAMID Technical Standard version
- Deployment/Component/Connection functional scope description (as above)
- Deployment/Component/Connection development stage (e.g. specification, design, or implementation)
- Security classification
- Compliance assessment result and rationale

A compliance declaration is intended to provide a consistent way to document the above information that will help Exploiting Programmes by:

- Supporting the ease of integration of components and deployments.
- Improving the ability to re-use components between differing deployments.
- Improving the ability to efficiently insert and update capability of a deployment, rapidly and with reduced risk.

E.2.4 Use of PRA Lexicon

The common language provided by the PRA can be used in compliance declarations to make it easier for Exploiting Programmes to compare existing components against their requirements. The PRA has been designed around a service modelling approach to system development and a similar approach to development of PYRAMID compliant software is recommended. For ease of reading, this guidance information has been written based on the assumption that a service modelling approach will be adopted by PYRAMID exploiters. Should an exploiter wish to adopt an alternative approach the principles outlined in this guidance would still apply with references to ‘services’ replaced by the appropriate artefact for the approach taken.

E.2.5 Deployment Programme Lifecycle

Development of a PYRAMID component can be considered within the process of a specific programme or system development. An Exploiting Programme should declare its compliance at its key programme milestones.

Showing how a component is PYRAMID compliant at differing stages of its lifecycle supports the programme’s design reviews, conducted at appropriate maturity points, ensuring the programme’s progress is consistent with the achievement of PYRAMID compliance. It is therefore anticipated that compliance reviews would be held as part of, or in support of, an Exploiting Platform’s design reviews.

Component compliance may be stated for a component at any stage of development. For example, compliance may be declared for platform independent component design, platform specific component design, or a delivered component product.

Similarly, the compliance for a deployment may be assessed and declared for any stage of development.

It is recommended that PYRAMID compliance assessment, for PYRAMID components, component connections and deployments as a whole, begins early in the development lifecycle and is maintained throughout the development. Any non-compliances can then be addressed at the earliest opportunity to minimise impact.

E.3 PYRAMID Component Compliance Supporting Information

E.3.1 PYRAMID Component Subject Matter

This section provides guidance on the rule, defined in the PYRAMID Technical Standard, Ref. [1], to determine if a PYRAMID component is compliant with the PYRAMID Technical Standard.

The basis of component compliance is conformance with the PRA defined subject matters, and thus, the absence of pollution.

Each PRA component represents a discrete area of subject matter. Subject matter pollution occurs when a PYRAMID component includes any subject matter of a PRA component other than the target PRA component. The PRA defines a set of responsibilities for each component that define and bound the functional scope of the component. The responsibilities for each component are defined in the PYRAMID Technical Standard, Ref. [1], section Component Definitions, and provide the normative aspect of the component definition for the purpose of component compliance assessment. Component compliance in respect of subject matter conformance is achieved by demonstrating that the functionality of a PYRAMID component is consistent with the scope of the responsibilities of the target PRA component.

The rule for component compliance is defined in the PYRAMID Technical Standard, Ref. [1] and duplicated in Table 26 below for convenience. The rule is applicable to all PYRAMID component variants, for example a PYRAMID component that implements only a subset of the PRA components responsibilities. Where a component is realised using extensions, the rule is applicable to the parent and extension components.

Rule Number	PYRAMID Component Compliance Rule
Component_rule_1	A PYRAMID component’s content shall be consistent with the responsibilities of the target PRA component.

Table 26: PYRAMID Component Compliance Rule

The different artefacts of the PRA component definition provide different views of a component’s subject matter, specifically the responsibilities, subject matter semantics and services. While the component responsibilities provide the normative element of a PRA component definition that form the basis of the compliance rule, each of these artefacts provides an aid to understanding a component’s subject matter:

- The component responsibilities define the functional scope that the component may provide within a system.
- The services give a detailed view of the functions provided and consumed by the component and the information at the component interfaces.
- The semantics give an overall view of the information and the concepts that the component reasons about and provides precise definitions that underpin the definition of component responsibilities and services.

Each of these three artefacts is discussed in the following sections. More information on subject matter can also be found in section PRA Design Principles.

E.3.1.1 PRA Component Responsibilities

The PRA defines responsibilities for each component, as specified in the PYRAMID Technical Standard, Ref. [1], section Component Definitions. For each PRA component, the complete set of responsibilities is defined by:

- The responsibilities defined in the Responsibilities section of the PRA component definition, which includes:
  - Responsibilities that are unique to a component's subject matter.
  - Responsibilities that are specialised versions of the generalised responsibilities defined in the Component Composition.
- Generalised responsibilities, defined in the Component Composition, that have not been specialised in individual PRA component definitions but are identified as being applicable to the component.

The set of a PRA component's responsibilities scope and bound the functionality that the component may provide within a system. The component compliance rule requires that the content of the PYRAMID component is consistent with the responsibilities of the target PRA component. Depending on the stage in the development lifecycle at which the compliance assessment is being made, this content may be expressed in the requirements, design or implementation of the PYRAMID component. At whatever stage the assessment is being made, the content may be deemed to be consistent with the responsibilities of the target PRA component if that content is necessary to fulfil an aspect of one or more of the target PRA component's responsibilities *and* is not content that should be provided by another PRA component.

Ensuring that this is achieved requires a sufficient breadth of knowledge of the scope of the PRA component set. For example, where one PRA component should have a dependency on a second PRA component, the functionality of both might be incorrectly developed as a single PYRAMID component if it were not recognised that the PRA defines two components with distinct subject matters. An in-depth knowledge of the whole PRA is not required to achieve this and in the first instance potentially relevant PRA components can be identified based on component name, role and overview. The subset of PRA components identified as potentially relevant to the area of capability under development can then be explored in more detail through a review of the component's defined responsibilities, semantics and services as necessary.

It should also be noted that for a PYRAMID component's content to be consistent with the responsibilities of the target PRA component does not require it to fulfil the full functional scope of those responsibilities. A component variant whose content is consistent with, but only partially fulfils, the responsibilities of the target PRA component may therefore be compliant.

It is also possible that a PYRAMID component incorporates content that does not appear to map directly to the responsibilities of the target PRA component but neither does it map to the responsibilities of another PRA component. In this case a component may still be assessed as being PYRAMID compliant if it is deemed that the included content of the PYRAMID component is consistent with the intended scope of the target PRA component subject matter, but this has not been fully captured or expressed within the PRA component's defined responsibilities. In such cases this should be observed on the compliance declaration and, if agreed, may be reflected in a subsequent revision of the PYRAMID Technical Standard.

Thus, in respect of compliance with the PYRAMID Technical Standard, the PRA component's defined responsibilities provide the normative content referenced in the component compliance rule in Table 26. Other elements of the PRA component definition and their relationship to the component responsibilities are discussed in the following sections.

E.3.1.2 PRA Component Subject Matter Semantics

The PRA defines the subject matter semantics for each component, as specified in the PYRAMID Technical Standard, Ref. [1], section Component Definitions. For each PRA component, this is defined in the Subject Matter Semantics section of the component definition.

The component's subject matter semantics defines the scope of the component by defining the entities (artefacts that model concepts that may or may not exist in the real-world) that the component reasons about. It provides precise definitions for these entities that help to scope the component responsibilities that reference these definitions. This section of the component definition also includes a semantics diagram, showing the relationships between entities. The relationships between entities provide an aid to understanding the scope of the component. This aspect of the subject matter semantics is not prescriptive however, and thus not normative in respect of compliance with the PYRAMID Technical Standard.

In respect of compliance with the PYRAMID Technical Standard, while the semantics provide detailed information on the scope of the concepts that the component reasons about, this information is normative only in the context of the component responsibilities, i.e. in providing precise definitions that help to scope the responsibilities.

E.3.1.3 PRA Component Services

The PRA defines provided and consumed services for each component, as specified in the PYRAMID Technical Standard, Ref. [1], section Component Definitions. For each PRA component, these are defined in the Services section of the component definition. Provided and consumed services are defined as follows:

- provided service: the means by which a component is asked to do something.
- consumed service: the means by which a component gets something done for it.

The service definitions are deliberately abstract to avoid being prescriptive about the specific details of the mechanism by which access to component functionality is achieved. The PRA component definitions do however include service definitions that provide detailed information on the nature of expected component services and an abstract view of the data needed to support the associated functions.

In respect of compliance with the PYRAMID Technical Standard, the component service definitions are informative and not normative. An exploiter is therefore not required to conform to the particular patterns, structure or format of these services. Neither will all the functions defined by these services be required for all Exploiting Programmes. However, the services that are developed for a PYRAMID component are required to be consistent with the target PRA component's defined responsibilities in accordance with the component_rule_1 (see Table 26). The following checks should be performed as good practice for assessing adherence to the PYRAMID Technical Standard and to reduce the risk of non-compliance:

1. Assessment of PYRAMID component services against target PRA component responsibilities.

   Are the PYRAMID component's services necessary to fulfil, or partially fulfil, one or more of the defined responsibilities for the target PRA component, and thus consistent with the subject matter of the PRA component? This check can be made for both provided and consumed services:

   - An assessment of provided services ensures that the functionality that a PYRAMID component makes available to the system has not extended beyond the responsibilities of the target PRA component.
   - An assessment of consumed services provides an additional check that what the component consumes from the system is consistent with what it is expected to consume to support the fulfilment of the target PRA component's defined responsibilities. The checking of consumed services may provide a key indicator of where a component has incorrectly internalised the subject matter of another PRA component.

   For the purpose of compliance assessment it may be beneficial to maintain traceability between a PYRAMID component's services and the target PRA component responsibilities that the service is fulfilling, or supporting the fulfilment of.
2. Assessment of PYRAMID component services against non-target PRA component responsibilities.

   Does the PYRAMID component define provided services that appear to be within the scope of the responsibilities defined for a non-target PRA component? Depending on the exact nature of the service being assessed, further analysis may be needed to determine whether the PYRAMID component has incorrectly incorporated functionality beyond its defined subject matter and is therefore not compliant with component_rule_1.
3. Assessment of PYRAMID component services against target PRA component services.

   Are the PYRAMID component's provided and consumed services within scope of the PRA component's provided and consumed services, respectively. A comparison of the PYRAMID component's services with those defined for the target PRA component provides a further means of checking adherence to the PYRAMID Technical Standard. While a PYRAMID component is not required to conform to the patterns, structure or format of the services defined for the target PRA component, comparing the former with the latter may provide an indication of non-compliance:

   - Where the scope of the provided or consumed services defined for the PYRAMID component go beyond those defined for the target PRA component, this may indicate that the scope of the PRA component's responsibilities has been exceeded.
   - Where the scope of the provided or consumed services defined for the PYRAMID component fall short of those defined for the target PRA component, this may simply reflect the fact that the PYRAMID component only incorporates a subset of the PRA

component's defined functionality. However, in the case of consumed services, the omission of a PRA defined service could also indicate that the functionality of another component has been incorrectly internalised and thus the scope of the PRA component's responsibilities has been exceeded, i.e. rather than the component including a consumed service to support a dependency on another component, the function on which the component depends has been incorporated within the component itself.

4. Assessment of PYRAMID component services against non-target PRA component service.

   Does the PYRAMID component define provided services that fully or partially overlap with the provided services defined for a non-target PRA component? This is a direct indication that the PYRAMID component has incorrectly incorporated functionality beyond its defined subject matter and is not compliant with component_rule_1.

E.3.1.4 Bridging Functions

The use of bridges is a key enabler to maintaining loose coupling between components, allowing components to be developed without knowledge of other components within the system. Bridges are used to perform the necessary translations that enable components to interact. Specifically bridges perform the following three functions, as described in Appendix A: PYRAMID Concepts, section Component Connections:

- Data type conversion (translating between the different formats and measurements used by different components).
- Semantic translation (translating the meaning of data in order to bridge the semantic gap between different component subject matter understandings), e.g. to translate a datalink message in to a component instruction.
- Invocation mapping between components (i.e. triggering behaviours in one or more components, as a result of event(s) generated in other components).

It is important that these bridging functions are not inappropriately embedded within PYRAMID components with the consequence that components become more tightly coupled, with a change in one component more likely to impact other components. Table 27 includes a number of additional guidance statements that are considered good practice to promote reuse, each of which is subsequently explained.

Guidance Statement Number	**PYRAMID Guidance Statement**
Guidance_statement_1	A PYRAMID component should not perform data type conversion or re-formatting, for the purpose of interacting with another PYRAMID component, except where this is the responsibility of the target PRA component.
Guidance_statement_2	A PYRAMID component should not perform semantic translation, for the purpose of interacting with another PYRAMID component, except where this is the responsibility of the target PRA component.
Guidance_statement_3	A PYRAMID component should not embed knowledge of the PYRAMID component architecture of a system (i.e. knowledge of specific components or which components provide which functions), except where this is necessary to fulfil the responsibilities of the target PRA component.

Table 27: PYRAMID Guidance Statements

Following this approach aims to ensure that the above bridging functions are not embedded within PYRAMID components, for the purpose of interacting with other PYRAMID components, except where the function in question is a legitimate responsibility of the target PRA component.

Explanation of Guidance_statement_1:

The purpose of this statement is not to impose a blanket rule on the inclusion of data type translations within components, but to highlight circumstances in which it may or may not be appropriate to do so. It may be appropriate to do so where:

- Components are being developed within a wider framework, or Exploiting Programme, for which the use of a common data model has been specified.

It may not be appropriate to do so where:

- The reason for the conversion is for the specific purpose of conforming to the design of another component with which the component needs to interact.
- The nature of the conversion needed warrants the use of the PRA Data Distribution component.

Explanation of Guidance_statement_2:

It is required that PYRAMID components adhere to the PRA defined subject matter. Components are therefore not expected to perform translations to the semantics of another component for the purpose of interacting with that component. This would result in component pollution and would be contrary to the component compliance rule in Table 26. It is important that where semantic translation is required this is undertaken by a bridge, or, where the nature of the translation needed is extensive or complex, the need for an implementation of the Semantic Translation component is recognised.

Explanation of Guidance_statement_3:

It is important that PYRAMID components are agnostic of the system within which they reside and do not inappropriately embed knowledge of the component architecture of a system except where this is consistent with the responsibilities of the target PRA component. PYRAMID components are therefore expected to express what they provide to, and require from, the system in their own terms as defined by their own subject matter.

For example, the design of component's consumed services should reflect the component centric view of what it requires from the system and not embed knowledge of where or how the system provides what it needs, i.e. the individual components that might satisfy any such dependencies. The structure and granularity of the component's consumed services should reflect the components own understanding of the nature of what it requires and it is the role of a bridge to map this to the provided services of the appropriate component or components within the system.

Exceptions occur where such knowledge is a legitimate aspect of the component subject matter, required to fulfil its responsibilities. Furthermore, in general, where multiple variants or instances of components exists within a system, components may need to maintain an understanding of peer components.

The purpose again of this approach is to promote component centric development that does not build in assumptions about the component architecture of the system within which the component is expected to operate. Such assumptions may undermine future growth and change within an

Exploiting Programme, and what may be valid for one Exploiting Programme may undermine reuse on another.

Exceptions:

There are exceptions to the above guidance that need to be taken in to consideration. In some cases the above bridging functions can be considered to be within the scope of the responsibility of a PRA components, for example:

- Data Distribution has responsibilities for the distribution and reception of data including the collation and formatting of data required for delivery.
- Semantic Translation has responsibilities for translating between the semantics of the participants involved in a data exchange.
- Information Brokerage has responsibilities that require it to understand the information needs of participants.

As can be seen, the scope of these PRA components overlaps with the functions of bridges. It has been recognised that while in some simple cases a bridge may perform the required translations, in others the required functionality is complex and exceeds the expected scope of a bridge. Where the potential for these complex cases to arise has been determined, the PRA has identified the associated subject matters and defined specific PRA components. These PRA components, Data Distribution, Semantic Translation and Information Brokerage are discussed further in Appendix A: PYRAMID Concepts, sections Data Exchange and Use of Communications. The use of these guidance statements therefore needs to recognise that for some PYRAMID components the above functions, typically performed by bridges, may be legitimately included within the PYRAMID component implementation.

E.3.1.5 PYRAMID Component Implementations

A PRA component can provide the basis of various PYRAMID component implementations, all of which may be compliant. The implementation will be dependent on the requirements of the deployment and not all aspects of a component, as defined in the PRA, may be required for a particular Exploiting Programme. For example, the following implementations are all potentially compliant:

- A component that implements a subset of the functional scope defined by the PRA component responsibilities.
- A component extension that implements a subset of the functional scope defined by the PRA component responsibilities.
- Multiple variants of a component that each implement a different or overlapping subset of the functional scope defined by the PRA component responsibilities.

Each of these implementations is potentially compliant with the PYRAMID Technical Standard, subject to the elements that are implemented being consistent with the responsibilities of the target PRA component in accordance with the component compliance rule in Table 26.

It should be noted that there are no compliance rules that apply specifically to component extensions other than the component compliance rule, which is applicable. The criteria that define an extension, and guidance on the use and benefits of extensions, is provided in Appendix A: PYRAMID Concepts, section Component Extensions. Where a component variant is developed that meets the criteria, the component can be declared as a component extension. It is expected that this information will also be recorded in a component compliance declaration (described in section The Importance of Compliance Declaration).

E.4 PYRAMID Component Connection Compliance Supporting Information

This section provides guidance on the rule, defined in the PYRAMID Technical Standard, Ref. [1], to determine if a component connection is compliant with the PYRAMID Technical Standard.

The basis of component connection compliance is the correct use of bridges to connect components together.

In addition to the rule for PYRAMID components, described above, it is also important that component connections are implemented appropriately. Bridges provide a mechanism for connecting a component to other system elements and are used to perform the translations necessary to enable components to remain independent of the structure and semantics of other system elements. However, it is important that bridge implementations do not inappropriately fulfil (or partially fulfil) the responsibilities defined for a PRA component. Rather, it should be recognised that the capability required falls within the scope of the responsibilities of a defined PRA component, and the appropriate PYRAMID component developed (based on this PRA component definition). Compliance is therefore achieved by demonstrating that bridges include only the necessary functionality to enable components to remain independent of knowledge of the structure and semantics of other system elements, without incorrectly incorporating functionality that is the responsibility of a PRA component.

Since the subject matter of some PRA components encompass some aspects of what are considered bridging functions, there are some exceptions that need to be taken in to consideration. These exceptions are discussed further below.

The rule for component compliance is defined in the PYRAMID Technical Standard, Ref. [1] and duplicated in Table 28 below for convenience. The rule for component connections applies to all connections between PYRAMID components. Note that bridges are not necessarily required between different instances or variants of PYRAMID components conforming to the same PRA component, or between a parent component and an extension of that component, since such interactions are internal to a single PRA component.

Rule Number	Component Connection Compliance Rule
Connections_rule_1	A bridge shall not fulfil a responsibility of a PRA component.

Table 28: PYRAMID Component Connection Compliance Rule

The design and implementation of component connections should be appropriate to the specific needs of the deployment. Section Bridging Functions explained how the implementation of PYRAMID components should seek to avoid unnecessarily coupling components by implementing bridging functionality within components, except where this is consistent with the responsibilities of the target PRA component. The converse of this is that a deployment should seek to avoid implementing the functionality of a PRA component within a bridge. Rather, it is important to recognise that the deployment solution may require the implementation of an additional PRA component. This ensures that an appropriate set of PYRAMID components is developed, within the defined PYRAMID deployment scope, avoiding the implementation of a potentially complex bridge with no reuse opportunity. The PYRAMID Technical Standard therefore includes the above component connection compliance rule to protect against this and to maximise the benefits of PRA compliant component development.

For similar reasons to those stated in section Bridging Functions, judgement is required in the assessment of component connection compliance with this rule. As noted in section Bridging Functions, a number of PRA components have responsibilities that overlap with the functions typically performed by bridges. It was also noted that in simpler cases it may be appropriate for such functions to be performed by bridges while in more complex cases the need for an additional PYRAMID component may be identified. Any component connection compliance assessment should therefore consider whether a bridge implementation has resulted in the inappropriate development of a complex bridge, that incorporates the functionality of a PRA component, and has failed to recognise (especially early in the lifecycle) the opportunity of developing an additional PYRAMID component with the associated benefits that brings.

Further information in relation to component connections can be found in the PYRAMID Technical Standard, Ref. [1], section Introduction to Component Connections and the PYRAMID Technical Standard Guidance document, Appendix A: PYRAMID Concepts section Component Connections.

E.5 PYRAMID Deployment Compliance Supporting Information

This section provides guidance on the rules, defined in the PYRAMID Technical Standard, Ref. [1], to determine if a PYRAMID deployment is compliant with the PYRAMID Technical Standard.

The basis of deployment compliance is the achievement of component compliance for all components within the PYRAMID deployment scope and a compliant means of connecting those components.

The rules for deployment compliance are defined in the PYRAMID Technical Standard, Ref. [1] and duplicated in Table 29 for convenience. The rules for deployment compliance apply to all components and component connections defined as being within the PYRAMID deployment scope.

Rule Number	PYRAMID Deployment Compliance Rule
Deployment_rule_1	All the components within the PYRAMID deployment scope shall satisfy the rules for PYRAMID component compliance.
Deployment_rule_2	All the component connections within the PYRAMID deployment scope shall satisfy the rules for PYRAMID component connection compliance.

Table 29: PYRAMID Deployment Compliance Rules

The concept of PYRAMID compliance at a deployment level enables a wider measure of compliance for a system development (or part thereof) beyond simply that of individual components. The relevant aspects of the system to which an assessment of PYRAMID compliance is determined to be applicable is defined by the PYRAMID deployment scope, as discussed in section PYRAMID Deployment Scope.

Having defined the PYRAMID deployment scope, the PYRAMID Technical Standard requires that all the PYRAMID components and component connections within the PYRAMID deployment scope are themselves PYRAMID compliant. Supporting information on these specific aspects of PYRAMID compliance is provided in sections PYRAMID Component Compliance Supporting Information and PYRAMID Component Connection Compliance Supporting Information.

E.5.1 Different PRA Baselines

While a deployment that uses components and component connections that align to different versions of the PYRAMID Technical Standard may still be compliant, it is recommended that exploiters attempt to use the same baseline version wherever possible, to minimise the risk of pollution due to changes in PRA component scopes between versions of the PRA.

Where a deployment does contain components or component connections that are compliant to different PRA baselines, it is advised that analysis is carried out to determine if there is subject matter pollution within the deployment, when assessed against the latest version of the PRA that has been used. This can be declared in the deployment compliance statement (described in section The Importance of Compliance Declaration), and provides useful information that supports future reuse or upgrade decision making and thus achievement of the PYRAMID compliance goal. For more information, see Appendix D: Deployment Guide, section Different Versions of the PRA.

Appendix F: Glossary

F.1 Introduction

This appendix provides the definition of a common set of terms and abbreviations relevant to both the PYRAMID Technical Standard, Ref. [1], and this document. There may be terms included within the glossary that are not used across both documents, but are only used in one of the documents.

F.1.1 Definition of Terms

Section Definitions of Terms provides a list of terms and definitions used within the PYRAMID Technical Standard, Ref. [1], and this document. The list includes PYRAMID specific terms and non-PYRAMID specific terms that are used throughout. References for any non-PYRAMID related sourced definitions have been included where appropriate.

F.1.2 Abbreviations and Acronyms

Section Abbreviations and Acronyms provides a list of abbreviations and acronyms used throughout the PYRAMID Technical Standard, Ref. [1], and this document.

F.2 Definitions of Terms

The table below defines two types of term:

PYRAMID specific:

These terms have a bespoke meaning within the PYRAMID Technical Standard and PYRAMID Technical Standard Guidance document. PYRAMID specific terms are indicated with their term descriptions highlighted in *italic* text.

Non-PYRAMID specific:

These terms are not bespoke to PYRAMID, however their definitions for use within the PYRAMID context are narrower in scope than their dictionary definitions. In addition to their definition, some terms also include additional information within their description to give the context of how the term applies within the PRA.

Term	**Description**
Achievability	*The ability to accomplish a requirement successfully.*
Action	*An activity defined in terms of what needs to be done. Actions are executed by coordinating resources.*
Activity	*Something that a system (or part of a system) does.*
Aircraft System	*An integrated system comprising of only one air vehicle and any number of supporting assets - such as its associated ground control station, mission planning systems and assets that are fully subordinate (such as missiles or support drones) to it or its control station. (Note that some subordinate assists may technically be air vehicles; however, they are an exception to the limit of one air vehicle. Likewise, it is possible for such assets to become or cease to be subordinate during the course of* a mission.*)*
Attribute	*An element of data that forms part or all of an interface on a* service.
Authorised Operator	Any person, user, or operator with validated credentials allowed to interact with a system to carry out a system role.
Availability	Property of being accessible and usable upon demand by an authorised entity. Ref. [30]
Bridge	*A mechanism for connecting a* component *to other system elements and that performs the translations necessary to enable the component to remain independent of the structure and semantics of other system elements.*
Catastrophic	Failure conditions that result in the death of one or more people. Catastrophic outcomes are considered DAL A within the PRA safety considerations.
Certification	The confirmation that the system complies with the applicable regulatory requirements (as agreed with the certifying authority, e.g. for airworthiness this is the MAA).
Component	*A piece of software or modelled artefact used as the basis for a piece of software (e.g. a* PYRAMID component *or* PRA component*).*
Component Behaviour	*Something that a* component *does in order to fulfil its responsibilities.*

Term	**Description**
Component Specification	*The precise requirement of the component as part of a specific* deployment*, described in terms of its* services *and/or entities.*
Confidentiality	Property that information is not made available or disclosed to unauthorised individuals, entities, or processes. Ref. [30]
Conflict	*A state where two or more demands (requirements or* constraints*) cannot be satisfied simultaneously.*
Constraint	*A limitation on the behaviour of a PYRAMID* deployment *at any level (whole system or constituent part).*
Consumed Service	*The means by which a* component *gets something done for it.*
Counterpart	*An abstraction (viewpoint), contained within a component, of a real-world object or concept that has a counterpart relationship with a different abstraction of the same real-world object or concept in another component.* *The abstraction can be for either a specific instance of the real-world object or concept (which may be represented as an object in object oriented software) or the concept from which a specific instance can be derived (which may be represented as a class in object oriented software).*
Counterpart Relationship	*The relationship between counterparts in different components, defining how they are related and how they may interact.*
Counterparting	*A concept whereby the same real-world object or concept is described from the viewpoints of different components.*
Critical	Failure conditions that result in major injury to people, loss of aircraft or a large reduction in safety margins. Critical outcomes are considered DAL B within the PRA safety considerations.
Cyber Attack	A deliberate and malicious exploitation of computer systems, technology-dependent enterprises and networks. Cyber attacks use malicious code to alter computer code, logic or data, resulting in disruptive consequences.
Data Driving	Data driving is the act of applying a data set to a component design, in order to provide a complete definition of how the component is required to behave.
Deployable Asset	*Any physical hardware (e.g. role fit* equipment*) carried on an* Exploiting Platform *that can be deliberately separated from the* Exploiting Platform *during a* mission*.*
Deployment	*A set of hardware and software elements forming a system (or part thereof) that satisfy the overall system requirements.*
Design Integrity	The extent to which the design is free from flaws that could give rise to or contribute to hazards, or to failure modes that contribute to a hazard. Ref. [32]
Dumb Asset	*A* deployable asset *that does not have a data interface with the* Exploiting Platform*.*
Equipment	*Hardware or a combination of hardware & software, that provides a capability or resource to the system under consideration.*
Executable Software	A computer file that contains encoded instructions capable of being executed by a processing unit. Executable software can be composed of one or more PYRAMID components.

Term	**Description**
Execution Platform	*The infrastructure supporting the execution, communication, etc. of application functionality, e.g.* ECOA, ARINC *653, Linux, Windows, and the computing hardware.*
Exploiter	*An organisation involved in the design and development of* PYRAMID components *or the design of PYRAMID* deployments*.*
Exploiting Platform	*A product (e.g. an air vehicle, ground station, or a test rig) that incorporates a PYRAMID* deployment*.*
Exploiting Programme	*A programme developing or incorporating PYRAMID components or a PYRAMID* deployment*.*
Extension Component	*A developed component that separates out or extends the functionality provided by a single* parent component*, by providing provided* services *that only the* parent component *has access to.*
Extension Point	*A* consumed service *that defines the* parent component*'s dependency upon an* extension component *for a single purpose.*
Extension Set	*A set of one or more* extension components *that satisfy the same* extension point*.*
Feasibility	*The practicality of achieving a solution.*
Flight	A collection of one or more aircraft, potentially of dissimilar types, performing roles and tasks to achieve the overall mission.
Flight Lead	The vehicle responsible for coordinating the activities of a flight to meet the objectives specified for the mission or supplied by the crew.
Flight Member	Any aircraft that forms part of a flight. Each member of the flight acts in support of the overall flight aims and of the other flight members.
Handover	*The process of performing a command and control transfer from one operator to another operator, e.g. between pilots on a twin-seat aircraft, operators on a single workstation, or across control stations.*
Health Data	Health data includes all data that is required as input for assessing the health and capability of the system. It includes, but is not limited to: • Hardware and software configuration data. • Fault and error codes (BIT reports). • Sensor data (including, but not restricted to, specific sensors for monitoring health and structural integrity). • System control, command and mode data. • Consumables data. • Usage data (for life and usage monitoring). • Manual measurements (requested and volunteered).
Integrity	(Safety context) The probability that the system will provide a specified level of safety. Ref. [11] (Security context) Property of accuracy and completeness. Ref. [30]
Item Development Assurance Level	The level of rigour of development assurance tasks performed on item(s). Ref. [27]
Logging	*The process of identifying and retaining data items associated with a component's processing that are not specific to its* subject matter*.*

Term	**Description**
Major	Failure conditions that result in minor injury to people, major damage to the aircraft or a significant reduction in safety margins. Major outcomes are considered DAL C within the PRA safety considerations.
Managed Resource	*A resource used by the system whose availability is limited and whose use by the system needs to be managed.*
Mission	*One or more aircraft ordered to accomplish one particular assignment.*
Mission Plan	The plan for the particular flight of one or more air vehicles from start-up / turnaround to shutdown / turnaround. It describes the planned flightpath and timings that the air vehicle should follow and the air vehicle's assigned mission objectives to be achieved in order to meet the tasking. A Mission Plan may be modified whilst an air vehicle is in flight as the result of dynamic re-tasking.
Mission Support System	A system which supports another system that is responsible for carrying out a real life operations (e.g. a mission planning system or a data extraction system).
Non-Repudiation	Ability to prove the occurrence of a claimed event or action and its originating entities. Ref. [30]
Objective	*A high level goal which either defines the purpose of the* mission *(e.g. the requirement to suppress enemy air defences) or is otherwise required of the* mission *(e.g. the requirement for aircraft survivability) that is assigned to the system to support a broader strategic goal.*
Parent Component	*A developed component that supports the use of* extension components*.*
Platform Independent Model	A representation of a system that is independent of the execution platform.
Platform Specific Model	A representation of a system that incorporates the execution platform.
Protection Domain	A grouping of components within a platform specific deployment context that have similar segregation requirements (e.g. for security, safety or specific functionality reasons) that are separated from other domains such that they are unable to interfere with the resources or processing in that domain. Communications between protection domains is strictly regulated.
Provided Service	*The means by which a* component *is asked to do something.*
PYRAMID Reference Architecture	*The open PYRAMID Reference Architecture is a set of platform independent component definitions for air system application software.*
PRA Component	*A PYRAMID reference artefact, defined by a role, a distinct set of responsibilities, entities and* services*, for a specific, discrete area of* subject matter*.*
PYRAMID Component	*A component that is intended to comply with a* PRA component *definition.*
Recording	*The process of identifying and retaining data items directly associated with a component's* subject matter*.*
Retention	The keeping of important data for future use or reference.

Term	**Description**
Retention Strategy	*A set of specific* retention *rules and supporting information covering which data is to be captured and retained and the conditions for* retention *(including duration, classification, etc.).*
Sanitisation	(Security context) The process of deliberately, permanently and irreversibly removing or destroying data to make it unrecoverable.
Security Accreditation	The formal, independent assessment of an ICT system or service against its IA requirements, resulting in the acceptance of residual risk in the context of the business requirement and information risk appetite. This will be a prerequisite for approval to operate.
Security Domain	A grouping of elements (e.g. components and data) with similar security requirements that are managed by a defined security policy such that groupings remain separate unless specific controls are in place (e.g. data encryption).
Security Enforcing Function	Function relating to specific controls that provide protection to the system (e.g. providing cryptography). Failure of a SEF could lead to a security breach.
Security Related Function	Functions that support the security activities within the system but are not directly involved in enforcing the separation of security boundaries or preventing cyber attacks. Failure of a SRF will not directly lead to a security breach but may diminish the system's ability to detect or counter a threat (e.g. security event logging).
Service	*The means by which a component is asked to do something, or by which a component gets something done for it. A service is formed of interfaces and activities.*
Situation Awareness	The perception of elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future. Ref. [31]
Storage	The action or method of storing data for future use.
Storage Media	The media used to store data.
Subject Matter	*The definition, semantics and behaviour associated with a topic or subject. This is used to define the bounded scope of a* PRA component*.*
System Integrator	*An organisation or individual involved in the wider integration of a PYRAMID* deployment*.*
Task	*The specification of a goal which needs to be achieved utilising the capability of an* aircraft system *(e.g. the need for an air vehicle to transit to a location, search an area, or for an aircraft system to disable a target).*
Trust	The confidence that a component or other system element will behave as expected.

Table 30: Definitions of PYRAMID Technical Standard Guidance Terms

F.3 Abbreviations and Acronyms

Name	Description
A/A	Air to Air
A/S	Air to Surface
AAR	Air-to-Air Refuelling
ACAS	Airborne Collision Avoidance System
ACID	Atomic, Consistent, Isolated, Durable
ADS-B	Automatic Dependent Surveillance – Broadcast
AEK	Algorithm Encryption Key
AMRAAM	Advanced Medium-Range Air-to-Air Missile
API	Application Programming Interface
ARINC	Air Radio Incorporated
ASRAAM	Advanced Short-Range Air-to-Air Missile
ATC	Air Traffic Control / Controller
ATIS	Automatic Terminal Information Service
ATS	Air Traffic Services
BIT	Built In Test
BS	British Standard
C2	Command and Control
CAT	Clear Air Turbulence
CBIT	Continuous Built in Test
CEP	Circular Error Probability
CIA	Confidentiality, Integrity and Availability
CIK	Cryptographic Ignition Key
COMSEC	Communications Security
CPDLC	Controller - Pilot Datalink Communications
CRC	Cyclic Redundancy Check
CRL	Certificate Revocation List
CSMU	Crash Survivable Memory Unit

Name	Description
CTT	Controlled-Trajectory Termination
CyDR	Cyber Defence and Risk
DAL	Development Assurance Level
DDS	Data Distribution Service
DEK	Data Encryption Key
DEW	Directed Energy Weapon
DME	Distance Measuring Equipment
DMZ	De-Militarized Zone
DoS	Denial of Service
EAL	Evaluation Assurance Level
ECM	Electronic Countermeasures
ECOA	European Component Oriented Architecture
EED	Electronic Explosive Device
EM	Electro-Magnetic
EMCON	Emissions Control
EMF	Electromagnetic Field
EN	European Standard
EO	Electro-Optical
ES	Electronic Surveillance
ESM	Electronic Support Measures
EUROCAE	EURopean Organisation for Civil Aviation Equipment
EW	Electronic Warfare
FAA	Federal Aviation Administration
FACE®	Future Airborne Capability Environment®
FPGA	Field-Programmable Gate Array
GDPR	General Data Protection Regulation
GMT	Greenwich Mean Time
GNSS	Global Navigation Satellite System

Name	Description
GPS	Global Positioning System
GSN	Goal Structuring Notation
GUI	Graphical User Interface
HF	High Frequency
HMI	Human Machine Interface
HMS	His Majesty's Ship
HOTAS	Hands On Throttle And Stick
HW	Hardware
IA	Information Assurance
IBIT	Initiated Built in Test
ICT	Information and Communications Technology
ID	Identifier
IDAL	Item Development Assurance Level
IDS	Intrusion Detection System
IEC	International Electrotechnical Commission
IFF	Identification Friend or Foe
ILS	Instrument Landing System
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
IP	Internet Protocol
IPS	Intrusion Protection System
IPSec	Internet Protocol Security
IR	Infrared
IRST	Infrared Search and Track
ISO	International Organisation for Standardisation
ISR	Intelligence, Surveillance and Reconnaissance
IT	Information Technology
IV	Interaction View

Name	Description
IVDL	Inter Vehicle Data Link
JPEG	Joint Photographic Experts Group
KEK	Key Encryption Key
LAR	Launch Acceptability Region
LDAP	Lightweight Directory Access Protocol
LDP	Laser Designator Pod
LIDAR	Light Detection and Ranging
LRU	Line Replaceable Unit
MASS	Master Armaments Safety Switch
MB	Megabyte
MBSE	Model Based Systems Engineering
MDA	Model Driven Architecture
MIKEY-SAKKE	Multimedia Internet Keying - Sakai-Kasahara Key Encryption
MIP	Multilateral Interoperability Programme
MITM	Man In The Middle
MSA	Minimum Safe Altitude
MSD	Minimum Separation Distance
MSS	Master Safety Switch
NATO	North Atlantic Treaty Organization
NIST	National Institute of Standards and Technology
O	Official
O-S	Official Sensitive
OMG	Object Management Group
OMI	Operator-Mission Interface
OSD	Office of Security of Defence
OTAR	Over The Air Rekeying
PBIT	Power Up Built in Test
PC	Personal Computer

Name	Description
PIM	Platform Independent Model
PMDH	Post Mission Data Handling
PNG	Portable Network Graphics
PRA	PYRAMID Reference Architecture
PRI	Pulse Repetition Interval
PRIME	Protocol Requirements for IP Modular Encryption
PSM	Platform Specific Model
QFE	Q code - pressure at airfield runway
QNH	Q code - pressure adjusted to mean sea level
QoS	Quality of Service
RA	Resolution Advisory
RCD	Residual Current Device
RCS	Radar Cross Section
RDP	Remote Desktop Protocol
RF	Radio Frequency
RFI	Request For Information
RoE	Rules of Engagement
RTB	Return To Base
RTCA	Radio Technical Commission for Aeronautics
RTPS	Real Time Publish Subscribe
S&RE	Suspension & Release Equipment
SA	Situation Awareness
SAE	Society of Automotive Engineers
SAM	Surface to Air Missile
SAR	Synthetic Aperture Radar
SATCOM	Satellite Communications
SC	Security Check
SCEO	Secret - Coalition Eyes Only

Name	Description
SDP	Session Description Protocol
SEAD	Suppression of Enemy Air Defences
SEF	Security Enforcing Function
SID	Standard Instrument Departure
SIEM	Security Information & Event Management
SIGMET	Significant Meteorological Information
SIP	Session Initiation Protocol
SNEO	Secret - National Eyes Only
SOA	Service Oriented Architecture
S.O.L.I.D.	Single-responsibility principle Open-closed principle Liskov substitution principle Interface segregation principle Dependency inversion principle
SOS	Store On Station
SOUP	Software of an Unknown Pedigree
SRF	Security Related Function
SSUN	Single Statement of User Need
SysML	Systems Modelling Language
TA	Traffic Advisory
TACAN	Tactical Air Navigation System
TB	Terabyte
TCAS	Traffic alert and Collision Avoidance System
TCP	Transmission Control Protocol
TDL	Tactical Data Link
TLS	Transport Layer Security
TOA	Terminal Operation Area
TRANSEC	Transmission Security
TS	Top Secret

Name	Description
TTP	Techniques, Tactics & Procedures
UAS	Uncrewed Air System
UAV	Uncrewed Air Vehicle
UC	Use Case
UCS	UAV Control System
UHF	Ultra-High Frequency
UK	United Kingdom
UML	Unified Modelling Language
US	United States
USB	Universal Serial Bus
UTC	Coordinated Universal Time
VoIP	Voice Over Internet Protocol
VOR	VHF Omnidirectional Range
VPN	Virtual Private Network

Table 31: Abbreviations and Acronyms