Software Engineering Institute
Carnegie Mellon University

Requirements and Architecture Specification of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System

Peter H. Feiler

December 2015

SPECIAL REPORT
CMU/SEI-2015-SR-031

Software Solutions Division

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

http://www.sei.cmu.edu

Copyright 2015 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

This report was prepared for the
SEI Administrative Agent
AFLCMC/PZM
20 Schilling Circle, Bldg 1305, 3rd floor
Hanscom AFB, MA 01731-2125

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This material has been approved for public release and unlimited distribution except as restricted below.

Internal use:* Permission to reproduce this material and to prepare derivative works from this material for internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions and derivative works.

External use:* This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

* These restrictions do not apply to U.S. government entities.

ATAM® and Carnegie Mellon® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM-0002957

Table of Contents

Executive Summary v

Abstract vi

1 Introduction 1

1.1 Background 1
1.2 Challenges in Current Requirements Documentation Practice 2

2 An Architecture-Led Requirements Specification Process 5

2.1 Context of an Architecture-Led Requirements Specification 5
2.2 ALRS and the FAA *Requirements Engineering Management Handbook* 6
2.2.1 Modeling Notations in Use 7
2.2.2 From State Variables to Information Flow 9
2.3 Improving Requirements Coverage 10
2.3.1 Elements of a System Specification 11
2.3.2 Coverage of Relevant Design and Operational Quality Attributes 11
2.3.3 Exceptional Conditions and Their Impact 13
2.3.4 Integrating Requirements Specification and Safety Analysis 14
2.3.5 Toward Incremental Lifecycle Assurance 15

3 Modeling ASSA as Operational Context for MIS and DCFM 17

3.1 Points of Interaction with the Operational Environment 17
3.2 Observed Entities and ASSA Context 18
3.3 Representing the ASSA Functional Architecture 21
3.3.1 Functional Elements of the ASSA System 22
3.3.2 The Concept of Tracks as Observation Representation 23
3.3.3 The Functional ASSA Architecture with Data Representations 25
3.4 Identifying the System Boundary 26
3.5 System Boundary and Roles of MIS 26
3.6 ASSA Functional Architecture Performance 29
3.6.1 ASSA System Response Time Analysis 29
3.6.2 Track Data Volume 30

4 ASSA System Design Architecture 32

4.1 ASSA Design Architecture 32
4.2 Performance Analysis of the ASSA System Design Architecture 34
4.2.1 Response Time Requirement Revisited 34
4.2.2 Interaction Protocols 34
4.2.3 Response Time Analysis 36

5 Maintaining Requirement Specifications in ALRS 38

5.1 The ReqSpec Notation 38
5.2 Goal and Requirement Specifications 39
5.3 ReqSpec Files 41
5.4 ASSA System Goal and Requirements Specification 44

6 Configurable Reference Architecture for Situation Assessment 47

6.1 Configurable Services and Interface Specifications 47
6.2 Reference Architecture Specification 48
6.3 Reference Architecture Analysis 50
6.4 Configuration for an Aircraft Platform 50

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

7 ASSA System Safety Analysis Approach 53

8 Summary and Conclusion 54

Appendix Acronym List 56

References 57

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

List of Figures

Figure 1: Quality of Requirements [NIST 2002] 2
Figure 2: Example of Stakeholder Requirements Documentation 2
Figure 3: Requirements Specification Across Architecture Hierarchy (Courtesy of M. Whalen) 3
Figure 4: Textual State Machine Specification 4
Figure 5: A Graphical State Machine Specification: Not a Requirement 4
Figure 6: Eleven Practices of the FAA Requirements Engineering Process [Adapted from FAA 2008a] 6
Figure 7: Four-Variable Model Mapped into an Architecture Specification 10
Figure 8: Constraints, Products, Resources, Elements, and Transformation (CPRET) 11
Figure 9: Operational Quality Attributes and Utility Trees 12
Figure 10: Fault Ontology and Its Application to a System Specification 13
Figure 11: Identification of Unsafe System Conditions 14
Figure 12: Identification of Contributors to Unsafe System Conditions 14
Figure 13: Identification of Safety System Functionality to Address Unsafe System Conditions 15
Figure 14: Requirement Decomposition and Verification 15
Figure 15: Requirement Decomposition and Verification One Layer at a Time 16
Figure 16: Incremental Assurance Through Virtual System Integration 16
Figure 17: Operational Context of the Situational Awareness System 18
Figure 18: Observables in the Operational Environment 18
Figure 19: Textual Threat Specification 19
Figure 20: ASSA Context as Model 20
Figure 21: Information Presented to Aircrew 21
Figure 22: ASSA System Functional Architecture 21
Figure 23: Situational Awareness Sensors 22
Figure 24: Data Correlation Specification 23
Figure 25: Track Type Definitions in AADL 24
Figure 26: Scope of Standard Track Representation 25
Figure 27: Interaction Inconsistencies in the ASSA Functional Architecture 25
Figure 28: System Boundary of DCFM 26
Figure 29: Identification of MIS Services as Layered Architecture 27
Figure 30: ASSA System Information Flows and the Data-Conversion and Data-Storage Services 28
Figure 31: Response Time for Observed Enemy Radar Track 30
Figure 32: Response Time for Assessed Enemy Radar Track 30

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Figure 33: ASSA System Design Architecture with SA Data-Conversion and Data-Storage Services 33

Figure 34: Three-Step Pull Protocol 35

Figure 35: Two Implementation Configurations of the Three-Step Protocol 36

Figure 36: Response Time Analysis Results with Latency Contributions for the Cross-Partition Pull Protocol 37

Figure 37: Sample of Imported Stakeholder Requirements 44

Figure 38: Goal Set for ASSA Sensors 45

Figure 39: Example of Requirement Specification Aligned with an AADL Model 45

Figure 40: Project with AADL Model Packages and ReqSpec Files 46

Figure 41: Configurable ASSA Service 47

Figure 42: Partial Service Specification 48

Figure 43: ASSA Reference Architecture 49

Figure 44: Aircraft-Specific Interface Configuration 50

Figure 45: Aircraft Sensor Configuration 51

Figure 46: ASSA Data-Conversion Service Configuration 51

Figure 47: Aircraft Configuration 52

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Executive Summary

The Carnegie Mellon University Software Engineering Institute (SEI) was involved in an Architecture-Centric Virtual Integration Process (ACVIP) shadow project for the U.S. Army's Research, Development, and Engineering Command Aviation and Missile Research, Development, and Engineering Center Science & Technology Joint Multi-Role program on the Joint Common Architecture (JCA) Demonstration. The JCA Demo used the Modular Integrated Survivability (MIS) system, which provided a situational awareness service that was integrated with two instances of a Data Correlation and Fusion Manager (DCFM) software component, which was contracted to two suppliers.

The purpose of the ACVIP shadow project was to demonstrate the value of using ACVIP technology, in particular the architecture models expressed in the Society of Automotive Engineering (SAE) Architecture Analysis & Design Language (AADL) standard, for discovering potential system integration problems early in the development process. To do this, the SEI team first captured information from existing requirements documents and other documentation as a requirements specification and architecture model expressed in AADL and a textual requirement specification subset of the draft Requirement Definition & Analysis Language Annex, referred to in this report as *ReqSpec*. We then analyzed this system model for potential system integration issues.

This report summarizes the approach taken to capture in AADL the requirements and architecture specification of the DCFM and its integration with MIS and to analyze the results. In this report, we refer to the resultant system as the Aircraft Survivability Situation Awareness (ASSA) system. By using an architecture-led approach to specifying requirements, the SEI team quickly identified a number of issues that, if not addressed, could result in system integration problems between MIS and DCFM. We documented these issues in a separate report [Feiler 2015a]. The issues include understanding the stakeholder goals for the ASSA, identifying the system boundary between MIS and DCFM, clarifying mismatched assumptions in the interaction between MIS and DCFM, and understanding the implications of architectural decisions on the system's ability to meet the requirements. Potential implications of architectural decisions include assumptions in the DCFM data model sequence diagrams that may hinder the system in meeting response time requirements and in additional calibration requirements for DCFM that may create unexpected latency and latency jitter that can introduce errors into the track data.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Abstract

The Carnegie Mellon University Software Engineering Institute (SEI) was involved in an Architecture-Centric Virtual Integration Process (ACVIP) shadow project for the U.S. Army's Research, Development, and Engineering Command Joint Multi-Role program in the Joint Common Architecture (JCA) Demonstration. The JCA Demo used the Modular Integrated Survivability (MIS) system, which provides a situational awareness service that was integrated with two instances of a Data Correlation and Fusion Manager (DCFM) software component, which was contracted to two suppliers. The purpose of the ACVIP shadow project was to demonstrate the value of using ACVIP technology, in particular the architecture models expressed in the Society of Automotive Engineering Aerospace Standard 5506 standard for the Architecture Analysis & Design Language (AADL), for discovering potential system integration problems early in the development process. To do this, the SEI first captured information from existing requirements documents in AADL and the draft Requirement Definition & Analysis Language Annex. Then, by using an architecture-led approach to capturing requirements and architecture specification, the SEI team quickly identified a number of issues that, if not addressed, could result in system integration problems between MIS and DCFM. The SEI's findings allowed contractor teams to address these issues early in system development.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

1 Introduction

The Carnegie Mellon University Software Engineering Institute (SEI) was involved in an Architecture-Centric Virtual Integration Process (ACVIP) shadow project for the U.S. Army's Research, Development, and Engineering Command Aviation and Missile Research, Development, and Engineering Center (AMRDEC) Science & Technology Joint Multi-Role (JMR) program in the Joint Common Architecture Demonstration. The JCA Demo used the Modular Integrated Survivability (MIS) system, which provides a situational awareness service that will be integrated with two separate versions of a Data Correlation and Fusion Manager (DCFM) software component. The DCFM was acquired via a Broad Agency Announcement (BAA) from two suppliers.

In the JCA Demo ACVIP shadow project, the SEI team captured requirements found in various MIS project documents using an architecture-led requirements and architecture specification process and following the ACVIP approach. In that process, the SEI team identified shortcomings in the existing requirements documents that, if not addressed, would result in potential system integration problems between MIS and DCFM.

This report summarizes the approach taken to capture the requirements and architecture specification of the integrated DCFM and MIS, referred to in this report as the Aircraft Survivability Situation Awareness (ASSA) system, in the SAE Architecture Analysis & Design Language (AADL). We also describe the resultant model and analyses supported by the model.

1.1 Background

To perform the architecture-led requirements specification task, the SEI team was provided with a February 2014 version of the MIS Stakeholder Requirements document and MIS System/Subsystem Specification (SSS) document, which contains system requirements. SEI used these documents to present a first set of issues, such as lack of indication of the number of tracks to be maintained by MIS, at the ACVIP Technical Interchange Meeting. The SEI team also received the JCA Demonstration BAA and the DCFM Data Model document at the Technical Interchange Meeting in early May 2014.

On June 15 and 17, 2014, the SEI team received April 2014 versions of the MIS Stakeholder Requirements document, the MIS SSS document, the Situational Awareness Data Service Software Design Description, the WeaponWatch Manager Software Design Description, the MIS System/Subsystem Design Description, the MIS SSS, and the DCFM BAA Supplement package. The SEI team also received July 3, 2014, versions of the MIS Stakeholder Requirements document, the MIS SSS, and a build plan, and July 17, 2014, versions of the MIS SSS and the MIS system model. On September 18 and 22, 2014, the SEI team received revisions of the MIS SSS and the DCFM data model.

The SEI team used the ACVIP approach to capture stakeholder requirements, system requirements, and the architecture design of the ASSA with its MIS and DCFM components. The ACVIP approach uses AADL and its architecture fault modeling and requirements specification extensions to represent relevant information in a single model with well-defined semantics. The tool

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

environment for AADL supports type checking, model consistency checking, and analysis capabilities. The SEI team developed the AADL model over a three-month period, including two 2-day working sessions with AMRDEC representatives. The SEI team presented its findings in a series of meetings, including the May 2014 JMR face-to-face working session and JMR meetings with the two contractor teams. In addition, the SEI team did a walk-through of the model as part of the *Modeling System Architectures with AADL* course, which was taught to the JMR team and JCA contractors on September 15–19, 2014.

1.2 Challenges in Current Requirements Documentation Practice

A 2009 industry study of current requirements engineering practice showed that text-based requirements specification is prevalent and that Microsoft Word® and DOORS® are the primary tools [FAA 2008b]. The result is that 70% of all software system problems are introduced during requirements specification and architecture design, while 80% are discovered after unit testing [NIST 2002, Redman 2010]. Figure 1 shows the top 5 of 12 quality issue categories characteristic of such requirement specifications [Hayes 2003]. It indicates that the quality of requirements can easily be improved by better requirements coverage.

Requirements error	%
Incomplete	21%
Missing	33%
Incorrect	24%
Ambiguous	6%
Inconsistent	5%

Figure 1: Quality of Requirements [NIST 2002]

Figure 2 shows a sample from the MIS Stakeholder Requirements document to illustrate the quality of the requirements. For example, the Mission Planning section indicates that MIS is expected to “enable” a set of capabilities. Note that a hardware processor could enable these capabilities. The Situational Awareness section suggests that MIS acts as integration mechanism for ASSA. However, a communication mechanism such as publish/subscribe can play that role. In other words, these requirements provide little insight as to what is specifically expected from MIS.

SR_59	**6 Mission Planning**
SR_73	MIS shall be compatible with Aviation Mission Planning System (AMPS) planning information
SR_74	MIS shall enable determining a primary route during pre-mission planning
SR_75	MIS shall enable in-flight mission planning
SR_76	MIS shall enable determining at least one alternate route during pre-mission planning
SR_77	MIS shall enable in-flight mission re-planning
SR_46	**7 Situational Awareness**
SR_64	***7.1 Location***
SR_78	MIS shall convey non-degraded aircraft location information from systems that are integrated through MIS
SR_79	MIS shall convey non-degraded airspace Situational Awareness (SA) information from systems that are integrated through MIS

Figure 2: Example of Stakeholder Requirements Documentation

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Often a collection of requirement statements spans multiple layers of an architecture hierarchy. This is illustrated in Figure 3 on a simple medical device example. On the left-hand side, four requirements statements for a patient monitoring system are shown. They are precise in that they provide details about volume and time. However, viewing the requirements statements in the context of an architecture representation reveals that they represent requirements at four different levels in the architecture hierarchy. In other words, the set of requirements makes assumptions about a (partial) system architecture.

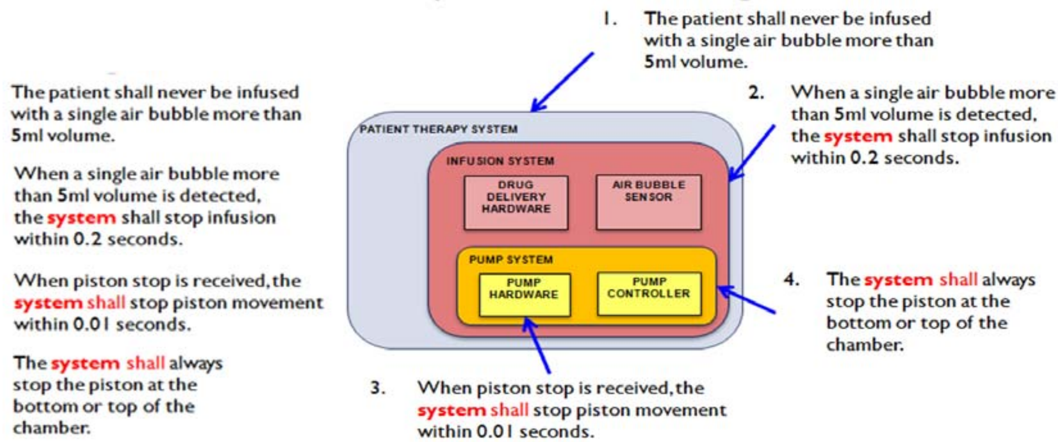

Figure 3: Requirements Specification Across Architecture Hierarchy (Courtesy of M. Whalen)

In the case of JMR, we encountered similar assumptions about system architecture in requirements statements. Many of the MIS requirements primarily described desired capabilities of the resultant ASSA system rather than MIS. Furthermore, two MIS components reside in a support layer below DCFM, while a health-monitoring component resides in a layer above DCFM (see Section 3.5).

In text-based requirement specifications, two rules are used to assess their quality:

- traceability to stakeholders and stakeholder requirements: For this reason, DOORS is a popular tool for managing requirements.
- the word “shall”: A statement or graphical presentation of a model in a requirements specification document is not considered a requirement if it does not include the word “shall.”

Figure 4 and Figure 5 illustrate these rules. They are taken from the MIS SSS. The documentation of the state machine describing the operational modes to be supported by MIS spans multiple pages. From this description, it is difficult to tell whether the specification of this state machine is complete.

The state machine in Figure 5 is not considered to be a requirement specification because it does not contain the word “shall.” However, it provides a more concise specification of the desired behavior and, as a model, it allows the specification to be processed by analytical tools. Even visual inspection provides a quick understanding of desired behavior. For example, a reader can see that the system does not have a transition that supports a reset operation to reinitialize the system after a shutdown. The system can be reset only by restarting the computer that hosts the MIS.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

ID	MIS System Requirements Specification	Requirement
SYS_51	**3.1 Required states and modes** Definitions: **“shall include”: How do we know there are only three states?** State: The system condition during a given time frame	False
SYS_52	The MIS System shall include the Startup system states.	True
SYS_634	The MIS System shall include the Operations system state.	True
SYS_635	The MIS System shall include the Shutdown system state.	True
SYS_54	After a state transition to a new system state, the MIS System shall log the new system state entry event.	True
SYS_55	After a fault in any state, the MIS System shall log the fault event.	True
SYS_275	The MIS System shall ignore state transition events if the event is not defined for the current system	True

30 “shall” statements later: A state transition and its trigger condition

SYS_482	If the duration of the Startup State exceeds an elapsed time threshold specified in system configuration data as a value between 1 minutes and 20 minutes and at least one AST interface completes initialization, the MIS System shall send a fault report to the Host Support System	True
SYS_65	If the duration of the Startup State exceeds an elapsed time threshold specified in system configuration data as a value between 1 minutes and 20 minutes and at least one AST interface completes initialization, the MIS System shall transition to the Operations State.	True

Figure 4: Textual State Machine Specification

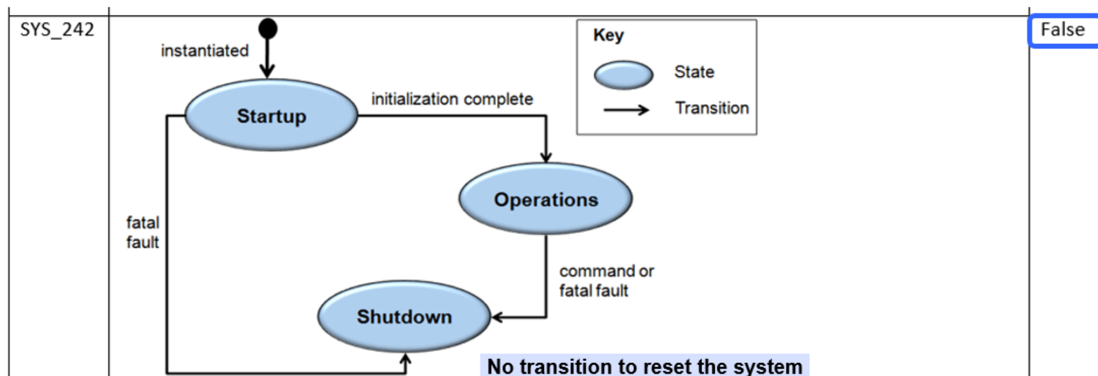

Figure 5: A Graphical State Machine Specification: Not a Requirement

These issues motivated us to develop an architecture-led approach to requirements specification, which we summarize in the next section.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

2 An Architecture-Led Requirements Specification Process

The objective of an architecture-led requirements specification (ALRS) approach is to specify requirements in the context of an architecture specification. There are several benefits to this approach:

- We can clearly relate requirement statements to specific elements in an architecture model.
- We gain insight into assumptions being made about a (partial) architecture.
- We can analytically verify an architecture design early in the process through virtual system integration.

There are three use case scenarios for the ALRS process:

- Existing stakeholder and system requirements documentation: In this case, we use the ALRS process to capture the information that has already been gathered in a text document. Capturing it as an architecture model annotated with requirements specifications allows us to identify ambiguities and conflicts in the original documents. This is the scenario guiding our shadow project. We summarize many of these issues in a separate report [Feiler 2015a].
- Negotiation of system requirements: A system requirements specification becomes a contract that a system provider will have to meet. Requirements must be specified in such a way that they are verifiable and conflicts between them have been resolved. Typically, a notional architecture design is used to assess whether the proposed system can satisfy the requirements. In other words, requirements affect architecture design decisions. We will demonstrate this interplay between requirements and architecture design when we examine the ASSA with the DCFM and MIS components in the context of a partitioned target platform that is based on the ARINC-653 specification, the avionics standard for partitioned time and space.
- Requirements elicitation from stakeholder: Stakeholder requirements elicitation involves creating a common understanding of the primary mission drivers, of the operational use-case (concept of operation) for the system, and of the boundaries between the system of interest, entities in its operational context, and its parts. We show how a model representation can help disambiguate and clarify domain concepts.

In addition, we utilize methods such as the SEI Mission Thread Workshop[1] and SEI Quality Attribute Workshop (QAW)[2] to guide the development of this common understanding and set priorities for stakeholder requirements.

2.1 Context of an Architecture-Led Requirements Specification

The *System Engineering Body of Knowledge* (SEBoK) under *Foundations of System Engineering* discusses stakeholder requirements and system requirements [BKCASE 2015]. For classification

[1] For an overview of the Mission Thread Workshop, see http://www.sei.cmu.edu/architecture/tools/establish/missionthread.cfm.

[2] For an overview of the Quality Attributes Workshop, see http://www.sei.cmu.edu/architecture/tools/establish/qaw.cfm.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

of requirements, it utilizes ISO/IEC/IEEE 29148 [ISO 2011]. Examples of classification of stakeholder requirements include service or functional, operational, interface, environmental, human factors, logistical, maintenance, design, production, verification, validation, deployment, training, certification, retirement, regulatory, environmental, reliability, availability, maintainability, design, usability, quality, safety, and security requirements. Stakeholders will also be faced with a number of constraints, including enterprise, business, project, design, realization, and process constraints.

This classification is similar to elements of a general model of sociotechnical control to analyze system accidents, originally developed by Rasmussen and adapted by Dr. Nancy Leveson of MIT for the Systems-Theoretic Accident Model and Processes (STAMP) method of accident causality analysis [Rasmussen 2000, Leveson 2012].

The ALRS process discussed in this report focuses on the specification of requirements for the system in its operational context, leading to a set of verifiable system requirements for the system and its subsystems. Integral to this process is the consideration of exceptional conditions that result in safety hazards or security vulnerabilities. ALRS assumes that requirements for the development process, as outlined in [BKCASE 2015], exist and are addressed.

2.2 ALRS and the FAA *Requirements Engineering Management Handbook*

The FAA *Requirements Engineering Management [REM] Handbook*, developed by the Rockwell Collins Formal Methods Group and published in 2009, illustrates 11 recommended practices that allow for verifying completeness and consistency of requirements analytically [FAA 2008a]. This handbook focuses on specifying system requirements systematically to make them verifiable. These 11 practices are shown in Figure 6. The handbook elaborates each practice in a number of substeps and illustrates their use with several examples. Dominque Blouin and colleagues have demonstrated how this 11-step process can be supported through a combination of the draft Requirements Definition & Analysis Language (RDAL) standard as annotations on AADL models [Blouin 2011]. They combine RDAL with use-case maps notation, a sublanguage of the International Telecommunication Union User Requirements Notation standard, to facilitate capture and validation of stakeholder requirements and their translation into system requirements [ITU-T 2008].

1. Develop the System Overview.
2. Identify the System Boundary.
3. Develop the Operational Concepts.
4. Identify the Environmental Assumptions.
5. Develop the Functional Architecture.
6. Revise the Architecture to Meet Implementation Constraints.
7. Identify System Modes.
8. Develop the Detailed Behavior and Performance Requirements.
9. Define the Software Requirements.
10. Allocate System Requirements to Subsystems.
11. Provide Rationale.

Figure 6: Eleven Practices of the FAA Requirements Engineering Process [Adapted from FAA 2008a]

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

In our case study, we focus on translating information found in existing documents into a model-based representation. In that process, we identify inconsistencies, ambiguities, and missing requirements information. In addition, we use the resultant model to virtually integrate MIS and DCFM and, in turn, analyze the resultant ASSA system to identify potential integration problems early in the development process.

2.2.1 Modeling Notations in Use

We use AADL, the Error Model Version 2 (EMV2) language, and a textual representation of the requirements specification subset of RDAL, referred to in this report as *ReqSpec*.[3] ReqSpec focuses on stakeholder and system requirements specification, while the full RDAL notation also allows users to specify a set of verification actions as elements of an assurance plan. Verification actions include model consistency checks, virtual integration analysis, and tests. Users can then track the execution of assurance plans by the results of verification actions throughout the system's lifecycle. See Section 2.3.5 for an outline of this incremental approach to lifecycle assurance.

We use AADL to

- specify the operational context of the system of interest (REM Practice 1). The MIS Stakeholder Requirements document provides a good description of the operational context of the ASSA system. The MIS SSS document provides a good description of the ASSA system functionality. We use the AADL abstract, system, and data component concepts to represent the elements of the operational context and the architectural structure of ASSA. We also use abstract feature, ports, and feature groups with connections to represent the interactions between these parts.
- identify the system boundary (REM Practice 2). We identify the DCFM as one of the functional components of ASSA. We identify MIS as consisting of three services: an SA data-conversion service and an SA data-storage service provided in a layer below the ASSA application layer, and an ASSA health-monitoring service provided in a supervisory layer above the ASSA application layer. We represent these layers in AADL.
- represent operational concepts (REM Practice 3). The use case scenarios fall into two categories:
  - developmental quality attributes such as portability through the use of the Future Airborne Capability Environment (FACE) and modifiability through the use of standard track representations
  - operational quality attributes such as situation assessment behavior, timely provision of assessment results (timing), and provision of valid assessment results (safety)
- represent environmental assumptions (REM Practice 4). We capture these assumptions as part of the interface specification of a system within the AADL model. See Section 2.3.1 for details.

[3] Note: The draft RDAL document defines a metamodel of the RDAL concept but not a textual representation.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

- develop the functional architecture (REM Practice 5). We do so for the ASSA system in an AADL model to fully understand the context of MIS and of DCFM. It helps us identify the desired functionality for DCFM and MIS.
- reflect implementation constraints (REM Practice 6). We reflect information from system design documents. We also reflect architecture constraints and decisions regarding the use of ARINC 653 partitioning. We use the AADL concepts of the virtual bus to represent system interactions as abstract protocols and the virtual processor to represent partitions. We then assess the impact of partitioning on end-to-end response time of critical flows.
- identify system modes (REM Practice 7). We record system modes as AADL modes.
- determine behavior and performance requirements (REM Practice 8). We use a prioritized utility tree to determine key operational requirements to be specified (see Section 2.3.2). We use flow specifications, data type representations, and various properties to capture the quality attributes of interest.
- capture safety requirements (implicit in REM practices). We capture safety-related information through the AADL EMV2 fault ontology and error propagation specifications. We provide the results of a full safety analysis in a separate report [Feiler 2015c].
- define software requirements (REM Practice 9). We map system functions into a task and communication architecture expressed by AADL threads, devices, and hardware platform specification. The threads represent executable software units. The specification includes properties addressing operational quality attributes.
- allocate system requirements to subsystems (REM Practice 10). As we elaborate the system architecture, we decompose system requirements to requirements on each subsystem. They are reflected in the AADL as properties and in the ReqSpec notation as explicitly traceable decomposition of requirement specifications that are directly associated with elements in the system architecture model.
- provide rationale. We record rationale as part of each ReqSpec-based requirement specification.

For the creation of AADL models, we utilize elements of the Virtual Upgrade Validation method [de Niz 2012]. The method helps users identify the type of system they are dealing with and the appropriate way of representing it in AADL. The method also provides guidance for focusing on common problem areas in software-reliant systems and ways to represent critical operational quality attributes.

We use AADL EMV2 Annex to

- systematically identify exceptional conditions that, when propagated to other systems and system components, represent hazards. We use the AADL EMV2 fault ontology as a checklist of these potential hazards. We use AADL EMV2 error propagation declarations to specify outgoing propagations of faults expected to be propagated and faults to be contained by the system (guarantees), and incoming error propagations of faults whose propagation from other components is acceptable or expected not to occur.
- specify failure modes, identify what component is responsible for detection of fault occurrences, and determine how the system responds to recovery actions.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

We use ReqSpec to

- distinguish between goals. System requirements specification acts as a contract between the customer and the system provider. However, stakeholder requirements may be ambiguously phrased, not verifiable, and in conflict with other goals and verifiable requirements.
- provide traceability to requirements specification and descriptions in existing documents.
- import a set of text-based requirement specifications into the ACVIP AADL model space. This allows us to access the documentation without external tools such as DOORS. It also allows us to record additional relationships between requirements, such as refinement, architectural decomposition, and evolution.
- associate requirement specifications with an architecture model expressed in AADL. This allows us to understand which system or subsystem is targeted by a requirement specification. It also allows us to identify gaps in coverage (see Section 2.3) and add requirements as appropriate.
- maintain consistency between the textual representation of a requirement specification and a representation of the requirement in the AADL, such as in the form of a property on a component or port.

For a summary of the notational capabilities of ReqSpec, see Section 5.1.

2.2.2 From State Variables to Information Flow

The practices described in the FAA *Requirements Engineering Management Handbook* draw strongly on the Software Cost Reduction method and the four-variable model originally proposed by Parnas and Madey [Parnas 1995] for specifying the requirements of the U.S. Navy's A-7E Corsair II aircraft [Schouwen 1990]. The four-variable model consists of *monitored variables* to represent observations in the physical system, *controlled variables* to represent control over the physical system, *input variables* as digital representation of monitored variables read by the software, and *output variables* as digital representation of controlled variables.

Later, the Software Productivity Consortium extended these ideas into the Consortium Requirements Engineering (CoRE) methodology [Faulk 1992, 1993], which was used to specify requirements for the C-130J aircraft. Many recommended practices on how to organize requirements are based on ideas originally developed with the CoRE method. The concepts of monitored and controlled state variables are also common in other specification methods for control systems, such as the State Analysis methodology, which is part of the NASA Mission Data System technology.[4]

Figure 7 illustrates how we map these variables into a flow-oriented architecture specification. Such a mapping is desirable as it reflects the actual information flow explicitly, rather than being hidden in the order in which different components perform read and write operations on the variables. A flow-oriented specification facilitates end-to-end flow analyses such as latency analysis. The mapping is simple and intuitive.

[4] For an overview of the NASA's State Analysis methodology, see http://mds.jpl.nasa.gov/public/sa.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

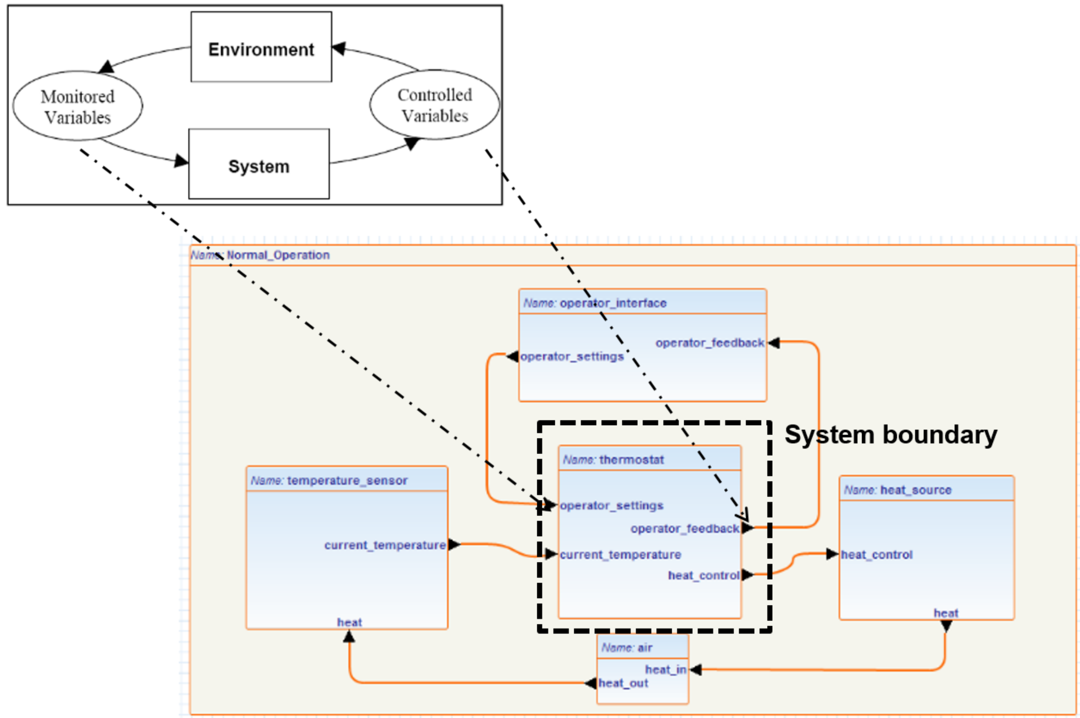

Figure 7: Four-Variable Model Mapped into an Architecture Specification

To reduce interaction complexity through global variables, it is a common modeling convention that these state variables are updated by one entity and can be read by multiple entities. Thus, the system updates the state variable that owns it and makes its value accessible to others through an outgoing port. Entities interested in the value of the variable receive that value through an incoming port. Port connections explicitly represent the data flow to all users of the variable content.

2.3 Improving Requirements Coverage

The ALRS captures stakeholder requirements as stakeholder goals for a system (borrowing the concept of a goal from goal-oriented requirements engineering) and captures system requirements as verifiable system specifications that act as contracts to be satisfied by system implementations. For that purpose, we introduce an analyzable textual notation called *ReqSpec* for expressing goals and requirements in the context of an architecture specification in AADL. A metamodel for this notation has been proposed as the RDAL standard annex for the SAE AADL (AS5506B) standard suite. In Section 5, we introduce this notation and offer guidance on using it with AADL to capture verifiable requirements and architecture specifications.

We improve the quality of requirements specification by providing a measurable way of assessing requirements coverage. Our method consists of three parts:

1. Identify all interaction points with the operational environment in terms of input–processing–output functionality and in terms of the resources and supervisory control necessary to provide this functionality. Each interaction point must be addressed by requirements.
2. Identify and quantify design and operational quality attributes that are key to achieving the mission. Each key quality attribute must be addressed by a requirement specification.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

3. Identify exceptional conditions that represent hazards to the safe and secure operation of the system. A fault ontology provides a checklist of failure conditions that are potentially propagated between system components as well as to and from the operational environment.

2.3.1 Elements of a System Specification

In addition, we utilize a framework for specifying a system that originated with the French Systems Engineering Society (Association Française d'Ingénierie Système). It is known as CPRET, which stands for Constraints, Products, Resources, Elements as input, and Transformations.[5] It is graphically illustrated in Figure 8. CPRET defines a system process to perform a set of transformations of input elements into products: respecting constraints, requiring resources, meeting a defined mission, corresponding to a specific purpose, and adapting to a given environment. The transformation is the sequence of ASSA functions illustrated in Figure 17.

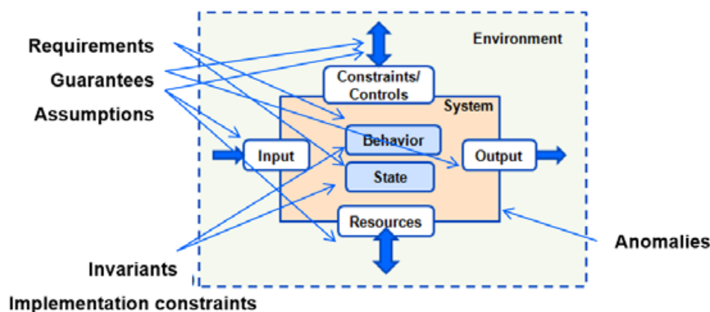

Figure 8: Constraints, Products, Resources, Elements, and Transformation (CPRET)

This framework helps us identify additional requirements and assumptions. In addition, it helps us identify resource requirements that ASSA has for the computing platform on which it executes and for other physical resources, such as the electrical power necessary to operate the sensors that are within ASSA. It also leads to a set of requirements for a command-and-control interface between the pilot and ASSA. As we elaborate the architecture of ASSA in the next section, it will help us recognize that the MIS health-monitoring service plays a supervisory role, while the MIS data-conversion and data-storage services play support roles by providing resources for accomplishing data interchange between components of ASSA.

Each interaction point must be addressed by requirements. The specification of each interaction point must indicate the type of interaction, the type of data or control being exchanged with others, the rate at which it is exchanged, and any exceptional conditions that the interaction must process. For input, supervisory control, and resource usage interaction points, these specifications represent assumptions about the operational environment. For output and supervisory control feedback, these specifications represent guarantees made by the system to others.

2.3.2 Coverage of Relevant Design and Operational Quality Attributes

Next, we utilize the concepts of quality attributes and utility trees from the SEI QAW and Architecture Tradeoff Analysis Method® (ATAM®). These quality attributes represent two categories of requirements:

---

5 For an overview of CPRET, see http://en.wikipedia.org/wiki/Process_(engineering).

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

1. developmental requirements, such as modifiability, portability, or assurability
2. operational requirements, which include the subcategories of mission-, safety- and security-critical requirements. Mission-critical requirements include function, behavior, and performance to complete a mission. Safety-critical requirements include function, behavior, and performance to mitigate loss leading to death, destruction, or damage. Security-critical requirements include function, behavior, and performance to mitigate compromise of classified or confidential information and loss of capability, resources, or security in any combination.

Figure 9 illustrates a partial set of quality attributes, three operational and one developmental. It also shows a refinement of the quality attributes into utility functions and their quantification into requirements whose satisfaction can be assessed measurably. The annotations of low, medium, and high (L, M, and H) pairs indicate levels of criticality and difficulty to help focus architectural design, evaluation, and verification. This utility tree becomes a checklist for assuring that requirement specifications address the relevant quality attributes of the system.

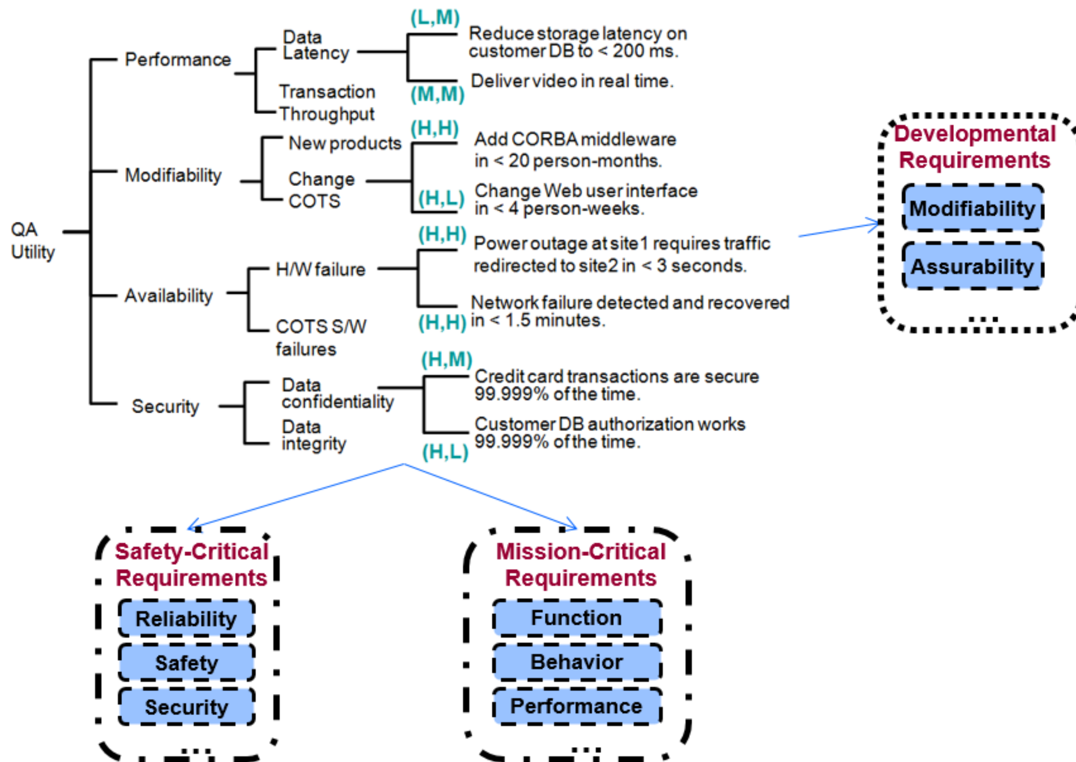

Figure 9: Operational Quality Attributes and Utility Trees

When specifying requirements for ASSA, MIS, and DCFM, we illustrate both developmental requirements and operational requirements. For example, developmental requirements might include portability, achieved by conforming to the FACE Standard; modifiability; and configurability, as a result of using standardized representation for observation tracks. Operational requirements include performance in terms of data volume and processing rates, response time in terms of end-to-end latency, and avoidance of unsafe conditions in terms of false positives and false negatives in situation awareness and timing discrepancies of time-sensitive information along multiple processing paths.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

2.3.3 Exceptional Conditions and Their Impact

Finally, we utilize a fault ontology that has been defined as part of the EMV2 language standard from the SAE AADL standard suite. The purpose of this extension to AADL is to support architecture fault modeling of various forms of system safety analysis.

Figure 10 illustrates the fault ontology on the left. This ontology focuses on error types that represent failure modes of systems being propagated to and from other systems. The most commonly used error type is omission, or the failure to provide a service or output. An example is failure to provide power. We distinguish between failure to provide a single item and failure of the source to provide any item. Commission occurs when a service or an output is provided at a time when it is not expected. Other error types represent value errors on individual items and sequences (sequence error), timing errors on individual items and sequences (rate error), replication errors for inconsistencies in replicated information or components, and concurrency errors for exceptional conditions when accessing shared resources. The terms used for the error types can be adapted to the domain. For example, omission may represent no power, and an above-range value error may represent a power spike.

The right-hand side of Figure 10 illustrates the systematic application of the error types to a system specification. It shows the interaction between a control system and a system under control. We can annotate this specification with error types to indicate whether certain types of exceptional conditions are expected to occur or not occur. Annotation of the interaction points represents assumptions and guarantees made by a system. Annotation of a connection indicates that the interaction itself is the source of an exceptional condition.

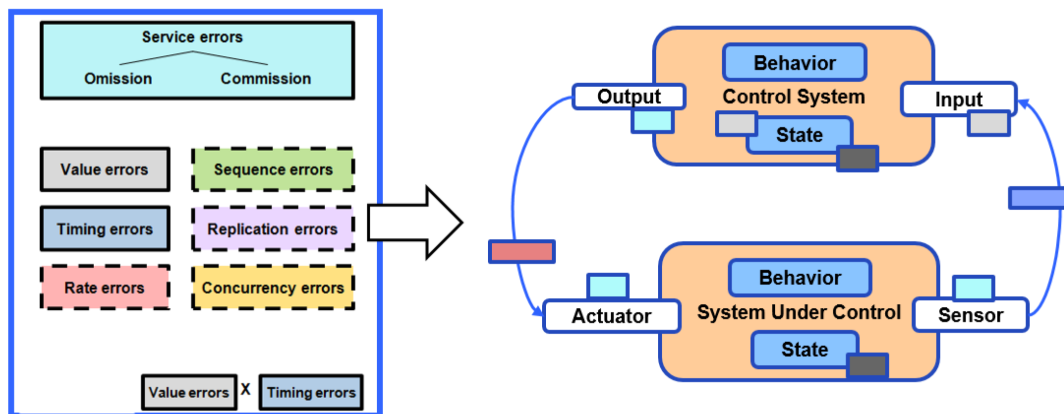

Figure 10: Fault Ontology and Its Application to a System Specification

The STAMP method has a similar model to classify hazards in control flows [Leveson 2012]. Some faults are characterized somewhat ambiguously (e.g., *inadequate* or *inappropriate*). These descriptions can be refined into more precise descriptions using the utility tree approach of the ATAM, leading to classifications that tend to align with the EMV2 fault ontology.

We apply the fault ontology at all levels of the system specification to ensure that the impact of exceptional conditions anywhere in the architecture design of the ASSA system are understood and addressed. This practice helps us minimize the potential for incidents due to occurrences of even “minor” faults.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

2.3.4 Integrating Requirements Specification and Safety Analysis

Including exceptional conditions in the requirements specification process facilitates the integration of safety analysis with requirements specification. We discuss the safety analysis of the ASSA, DCFM, and MIS systems in a separate report [Feiler 2015c]. Here we provide a summary of the three major steps involved in addressing safety hazards:

1. Identify unsafe system states that must be addressed as safety requirements (Figure 11). These states can be the result of failure modes of individual system components or unexpected interactions between components that in themselves are not failing.
2. Identify contributors to reaching unsafe system states; these exceptional system conditions lead to hazards and must be managed (Figure 12). Typically, they take the form of control actions that lead to unsafe system states or lack of control actions to recover from unsafe system states.
3. Identify capabilities for a safety system (derived safety requirements) in order to manage the exceptional conditions (Figure 13). These capabilities typically take the form of sensors to observe systems that are involved in unsafe system states and to detect these unsafe system states. Other derived requirements include restrictive conditions on control actions and additional control actions to provide reporting and recovery from unsafe system states.

Figure 11 through Figure 13 illustrate these steps in a simple train example.

Identify unsafe system states
- Failure hazards & interaction hazards (error sources)
- Exceptional conditions that contribute to incidents/accidents

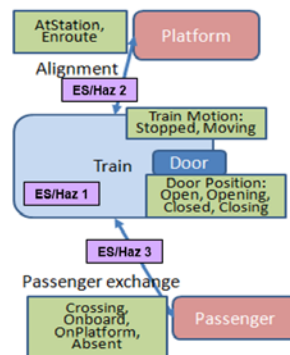

Figure 11: Identification of Unsafe System Conditions

Identify contributors to unsafe system states
- Control functions as error sources that result in
  - exceptional condition/unsafe state
  - inability to recover from exceptional condition

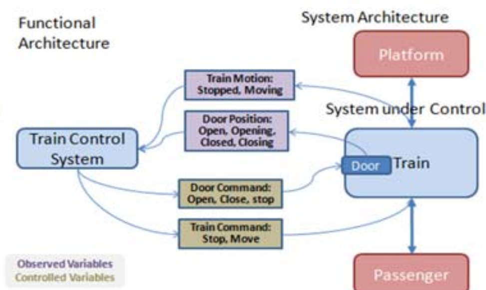

Figure 12: Identification of Contributors to Unsafe System Conditions

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

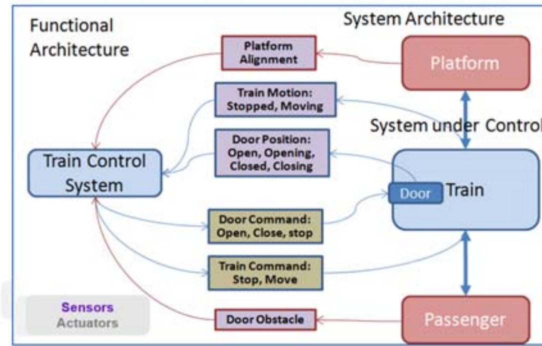

Figure 13: Identification of Safety System Functionality to Address Unsafe System Conditions

2.3.5 Toward Incremental Lifecycle Assurance

An objective of ALRS is to support an incremental lifecycle approach to safety-critical system assurance. The ALRS process addresses the gap that currently exists in many software-reliant system developments between system requirements and software requirements [Boehm 2006]. The ALRS process is driven by the close interaction between requirements specification and architecture design, as we described in Section 2.3.4 on safety analysis and derivation of safety system requirements. Similarly, we will refine the specification of ASSA into an ASSA system architecture that will become the context for identifying the system boundaries of DCFM and MIS and for determining the requirements placed on these subsystems by the ASSA as operational context.

Figure 14 illustrates generically how requirements associated with a system specification are decomposed into requirements of the subsystems once the first layer of the system architecture has been designed. In this context, we can assure that all system-level requirements are appropriately mapped into requirements for the subsystems. We also assure that additional requirements on each subsystem are addressed, such as a subsystem's use of the system's internal resources or exceptional conditions introduced by a subsystem that may or may not have been specified at the system level.

The right-hand side of Figure 14 illustrates a second step in this process: the association of verification activities with requirements as part of an assurance plan and their incremental execution throughout the lifecycle. In that context, we assess whether satisfying the subsystem requirements is sufficient evidence for meeting a system-level requirement or whether an explicit verification activity is necessary.

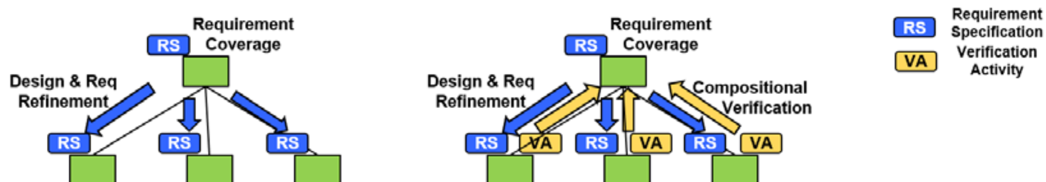

Figure 14: Requirement Decomposition and Verification

Figure 15 shows this process applied recursively. Because the hierarchy reflects the architecture abstraction, we may be able to perform compositional verification. A verification activity may op-

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

erate on the assumption that the subsystem requirements have been met, allowing for compositional verification, or it may provide results of a different level of fidelity, depending on the level of expansion in the architecture hierarchy.

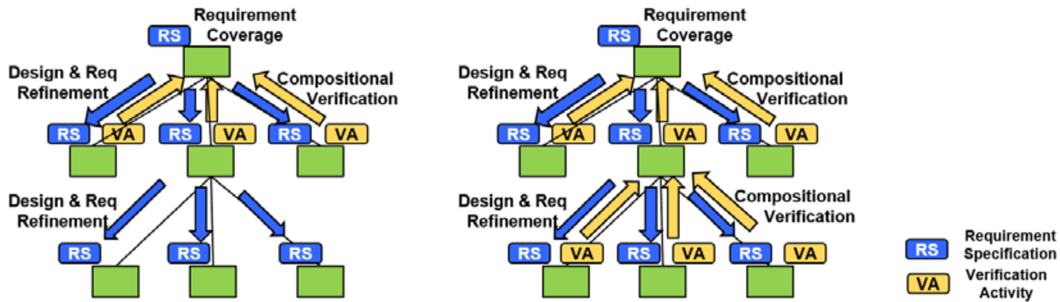

Figure 15: Requirement Decomposition and Verification One Layer at a Time

This process of incrementally refining a system architecture and analyzing the virtually integrated system early in the lifecycle leads to early discovery of system-level problems that are currently not discovered until after unit testing. The same process also leads us to assure the system incrementally throughout the lifecycle by evolving the assurance plan and incrementally executing verification activities against it. This is graphically illustrated in Figure 16.

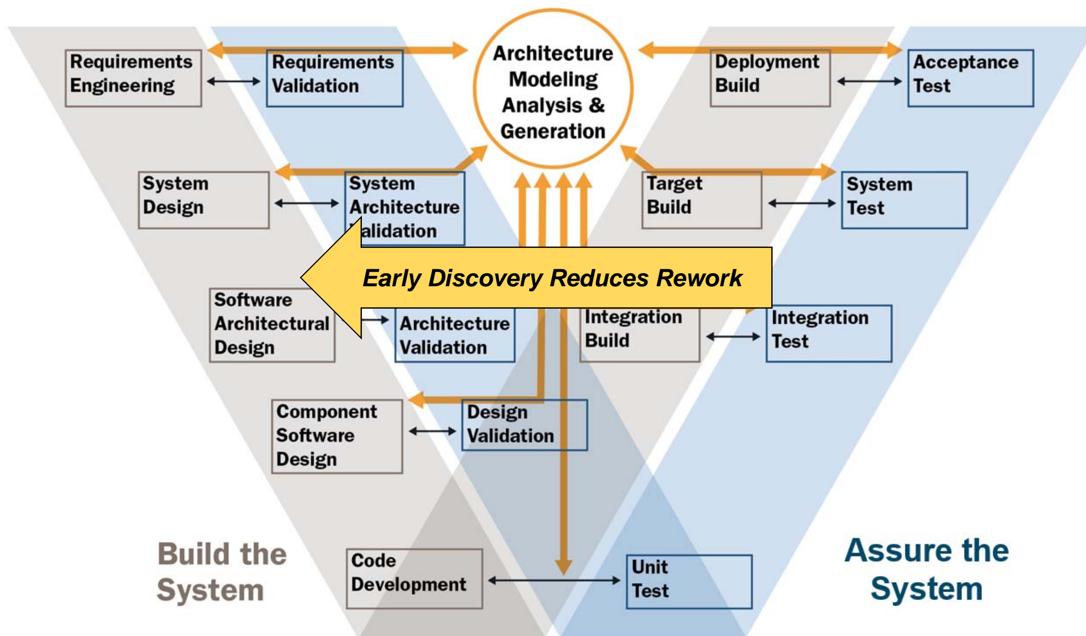

Figure 16: Incremental Assurance Through Virtual System Integration

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

3 Modeling ASSA as Operational Context for MIS and DCFM

Together, the MIS Stakeholder Requirements document, the MIS SSS document, and the BAA Supplement document provide descriptions of the operational context for MIS and DCFM. In this section, we explain how we capture this operational context in AADL models to help us identify the system boundaries of MIS and of DCFM.

To do this, we created three projects in the Open Source AADL Tool Environment (OSATE). The first project, *SituationalAwarenessCommon*, contains AADL models of concepts and services used in the other two projects. The second project, *SituationalAwarenessSystem*, contains the AADL model and requirement specifications as a record of the information found in the documentation provided to us. The third project, *SituationalAwarenessRefArch*, contains an AADL model of the ASSA reference architecture and its configuration for a specific aircraft platform.

3.1 Points of Interaction with the Operational Environment

In our case study, the first step is to translate the information from the MIS Stakeholder Requirements document into an AADL model and annotate it with appropriate specifications of stakeholder goals. This document tells us that we are dealing with a situational awareness system for aircraft survivability.

One objective of the stakeholder requirements elicitation process is to establish a common understanding of the application domain, domain-specific concepts, and desired capabilities. The ASSA system collects observational data about the operational environment, in particular about threats, obstacles, terrain, and weather. All these factors potentially affect the survivability of the aircraft. The ASSA system performs data correlation, data fusion, and situation assessment and reports the results to the pilot and an automated rerouting system. Both of these entities can take corrective actions to avoid or recover from situations that negatively affect aircraft survivability. In other words, ASSA together with the pilot or automated rerouting system acts as a control system to manage the flight path of the aircraft. For more information about sensor correlation and fusion processes, the Air University New World Vistas volume on sensors provides a nice framework [AU 1996].

We specify the entities of the operational environment as abstractly as possible, but still with a precise characterization relevant to performing situational awareness functions. For example, different threats may be characterized as stationary or moving, while obstacles may always be characterized as stationary.

Figure 17 illustrates the ASSA system in its operational context. The key entities that the system will observe are shown on the left as solid-line rounded rectangles, while additional entities are shown as dashed-line rounded rectangles. The figure also shows specific types of threats to be observed.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

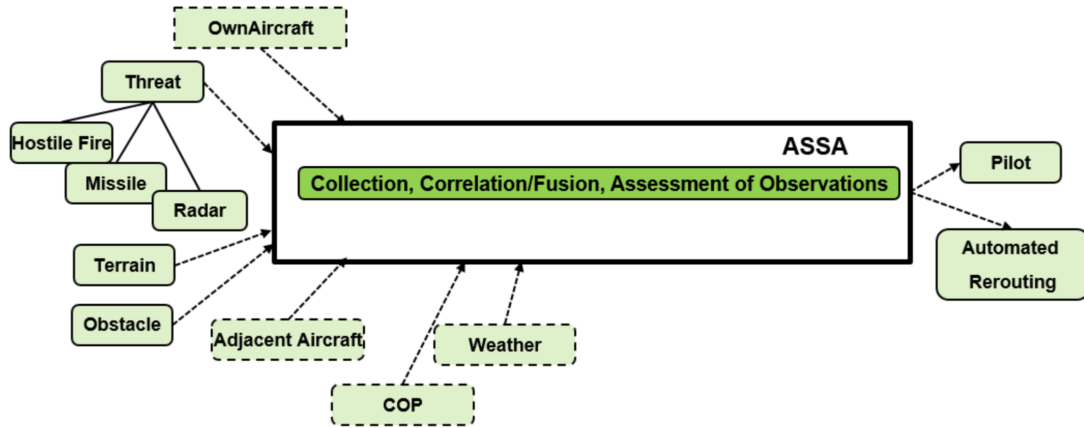

Figure 17: Operational Context of the Situational Awareness System

3.2 Observed Entities and ASSA Context

Another early step is to identify entities in the operational context that ASSA must track. We use data modeling techniques to capture those entities. With AADL, we can use the abstract component type construct to represent these entities. We place all the abstract component types in a package called *SAObservations*. For data modeling, we could also use other notations such as Unified Modeling Language (UML) class diagrams. In that case, we would map relevant aspects of the resulting UML data model into AADL using the Data Modeling Annex guidance.

The MIS Stakeholder Requirements document identifies three types of entities: threats, obstacles, and terrain. We use the *AVCIP::aliases* property to keep track of different terms that are used in various documents and seem to describe the same concept or entity. By recording this information, we can later confirm whether this is actually the case or whether each term represents a different concept or entity.

Threats are entities that can inflict damage to the aircraft with weapons. The stakeholder requirements and other documents identify different types of threat. We capture them as abstract component types in a type hierarchy, as shown in Figure 18. This is done by declaring a type as an extension of the type *Threat*, as shown in textual AADL in Figure 19. The type *Threat* indicates that all subtypes of threat must be considered. In some documents, only subsets of the threat types are identified; for example, the DCFM data model has an enumeration type that identifies only three of the threat types. The type hierarchy helps us recognize and resolve such ambiguities.

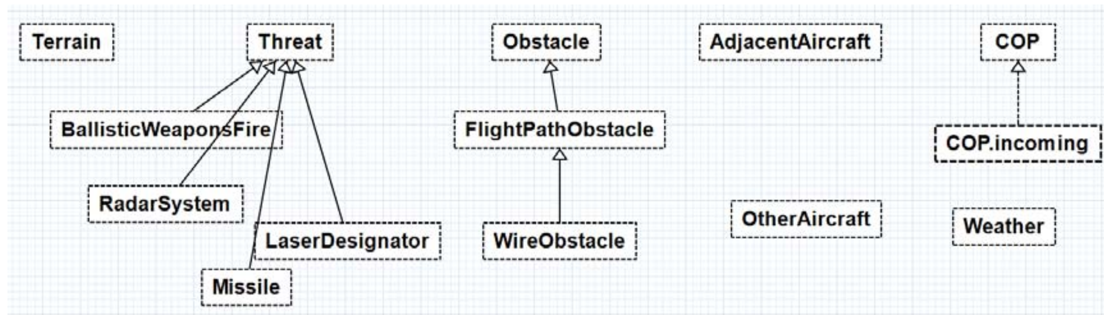

Figure 18: Observables in the Operational Environment

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

```
abstract Threat
end Threat;
abstract BallisticWeaponsFire extends Threat
properties
        ACVIP::aliases => ("HostileFire (in DCFM Data Model)",
        "HostileFireDetectionSystem");
end BallisticWeaponsFire;
```

Figure 19: Textual Threat Specification

Obstacles are entities that may be in the flight path and can cause a collision. Examples of obstacles are towers and electrical power lines. We find two more specific terms for obstacles: flight path obstacles and wire obstacles. By relating them in a component type hierarchy and providing appropriate descriptions, we can clarify the intended meaning and relationship of each term. For example, is *WireObstacle* a subclass of *FlightPathObstacle*, or are they two orthogonal categories?

One of the documents indicates that aircraft are lost more often due to collision with obstacles than due to threats. This provides a rationale in the stakeholder requirements for tracking obstacles. We initially record this information as part of the *Obstacle* abstract type using the *ACVIP::Rationale* property and include it as rationale for a stakeholder goal regarding obstacles.

The third type of entity is terrain. In low-altitude flight, it is critical for an aircraft and the pilot to be aware of the terrain.

Various documents also refer to other aircraft and adjacent aircraft, the weather, and a Common Operational Picture (COP) as entities that the system must be aware of. Again, by relating these entities we can reduce ambiguity in the interpretation of these concepts. It helps answer questions such as “Is weather part of the COP?” “Are other aircraft and adjacent aircraft the same?” and “Are enemy aircraft and coalition aircraft being distinguished?”

Finally, we specify the ASSA system as an AADL system type with two outputs, as shown in Figure 20. It is defined in the package *ASSASystem*. The two outputs are presenting situation awareness results to the aircrew and passing the results to an automatic control system, such as a flight path rerouter. We use the property *JMRMIS::ObservedObjects* to indicate the entities that ASSA will observe. The values refer to the abstract types we defined in *SAObservations*. This leaves open the decision as to whether ASSA sensors are part of the ASSA system or outside the ASSA system. The property *JMRMIS::ObservationRadius* is used to indicate the expected radius within which awareness is raised to the aircrew (according to the Stakeholder Requirements document).

As we identify information that is expected to be supplied to the ASSA system by other aircraft systems, we specify corresponding incoming features in the *ASSASystem* system type, such as *OwnAircraftPosition*. We use feature group declarations in cases where multiple pieces of information will be communicated, such as the information presented to the aircrew. Where appropriate we may identify a specific data type, as illustrated for *OwnAircraftPosition*.

```
system ASSASystem
features
```


Distribution Statement A: Approved for Public Release; Distribution is Unlimited

```
  IncomingCOP: in data port;
  FENNLocations: in data port;
  GeospatialData: in data port;
  EnvironmentalInformation: in data port;
  Weather: in data port;
  OwnAircraftPosition: in data port MissionSystemDataTypes::Position;
  AMPSInterface: in out event data port;
  ThreatAlerts: out feature;
  ASSAAirCrewPresentation: out feature group ASSAInterfaces::AirCrewSAInformation;
  ASSAAutoControl: out feature group;
properties
  JMRMIS::ObservedObjects => (
    classifier (SAObservations::Threat),
    classifier (SAObservations::Obstacle),
    classifier (SAObservations::Terrain) );
  JMRMIS::ObservationRadius => 5 NM applies to SA_AirCrewPresentation;
end ASSASystem;
```

Figure 20: ASSA Context as Model

Figure 21 illustrates how we use a feature group type declaration to specify the type of information expected to be presented to the aircrew. It is defined in the AADL package *ASSAInterfaces*. Where appropriate, we use properties to characterize the presented information, such as an indication of the observation radius covered by each data item.

```
feature group AirCrewSAInformation
features
  AircrewSphericalTerrainInformation : out feature {
    JMRMIS::ObservationRadius => 5 NM;};
  AircrewTerrainLocationAwareness : out feature {
    JMRMIS::ObservationRadius => 2 NM;};
  AircrewTerrainHeightAwareness : out feature {
    JMRMIS::ObservationRadius => 2 NM;};
  AircrewTerrainHazards : out feature;
  AircrewSphericalObstacleInformation : out feature {
   JMRMIS::ObservationRadius => 5 NM;};
  AircrewRelativeFlightPathObstaclePositionAwareness : out feature {
    JMRMIS::ObservationRadius => 5 NM;};
  AircrewFlightPathObstacleHeightAwareness : out feature {
    JMRMIS::ObservationRadius => 5 NM;};
  AircrewFlightPathObstacleSeparationAwareness : out feature {
    JMRMIS::ObservationRadius => 5 NM;};
  AircrewRelativeWireObstaclePositionAwareness : out feature {
    JMRMIS::ObservationRadius => 5 NM;};
  AircrewFlightPathObstacleHorizontalSeparationAlert : out feature {
```


Distribution Statement A: Approved for Public Release; Distribution is Unlimited

```
    JMRMIS::ObservationRadius => 5 NM;};
  AircrewFlightPathObstacleVerticalSeparationAlert : out feature {
    JMRMIS::ObservationRadius => 5 NM;};
  AircrewSphericalAdjacentAircraftInformation : out feature {
    JMRMIS::ObservationRadius => 5 NM;};
  AircrewRelativeAdjacentAircraftPositionAwareness : out feature;
  AircrewOtherAircraftPositionAwareness : out feature;
  AircrewOtherAircraftAltitudeAwareness : out feature;
end AirCrewSAInformation;
```

Figure 21: Information Presented to Aircrew

3.3 Representing the ASSA Functional Architecture

The MIS SSS document provides details about the ASSA functional architecture and system architecture in terms of ASSA sensors. These are illustrated in Figure 22, which shows several types of sensors representing the data collection functionality, several instances of data correlation and fusions functionality, one instance of situation awareness functionality, and presentation functionality. In a later section, we present a reference architecture for ASSA with each of these functional areas that then get instantiated for particular aircraft platform and sensor configurations.

Figure 22 shows which sensors are responsible for observing which threats, obstacles, and terrain. It also shows that the source of the own aircraft position is an embedded global positioning system (GPS)/inertial navigation system (EGI) external to ASSA. We see that a data correlation and fusion service combines tracks from two sensor sources, while a data correlation service deals with only one sensor source. Correlation in this context relates the position of observed entities to the own aircraft position. The resultant tracks are then assessed for situational awareness against proximity conditions for raising awareness and alerts.

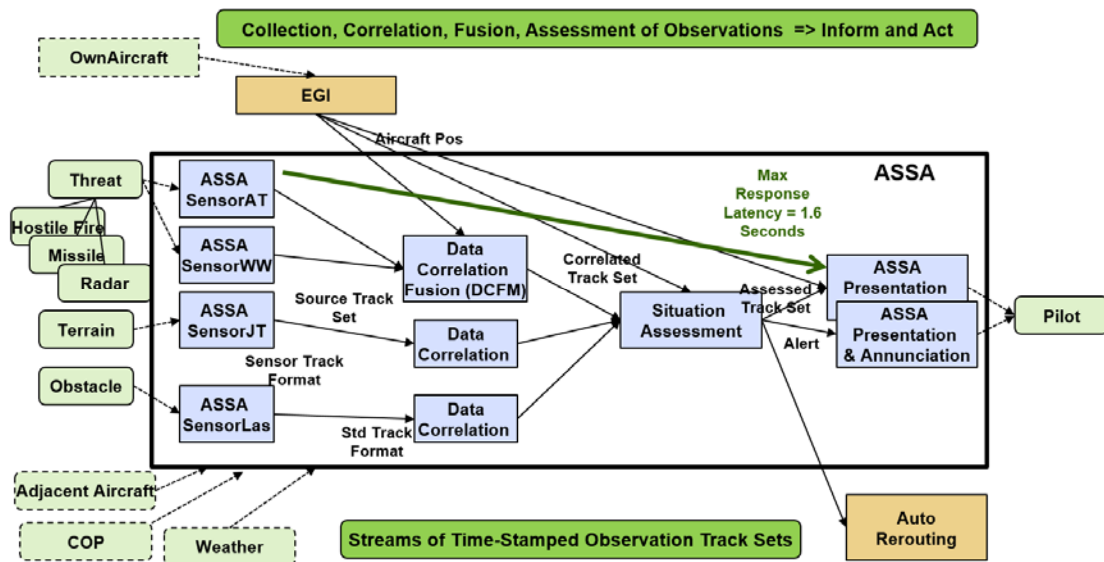

Figure 22: ASSA System Functional Architecture

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

3.3.1 Functional Elements of the ASSA System

We use the *ASSASensor* package to specify the different ASSA sensor types, shown graphically in Figure 23. The top row shows the four types of aircraft survivability threat sensors identified in the Stakeholder Requirements document. This document also distinguishes between active and passive sensors. The concrete sensors are represented by the AADL device component, while the concepts of *Sensor* and *ASSASensor* are represented as abstract components.

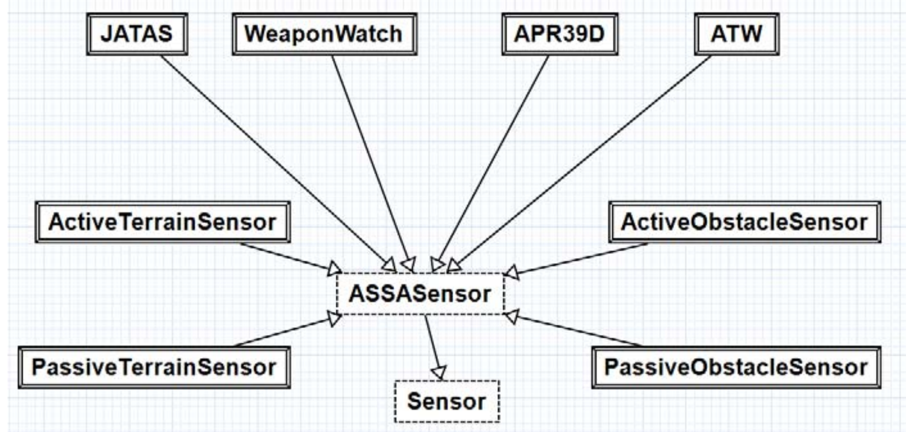

Figure 23: Situational Awareness Sensors

Each sensor type is characterized with a set of properties. They include project-specific properties and predeclared properties for devices:

- *JMRMIS::ObservationRadius*: the maximum radius within which the sensor observes entities
- *JMRMIS::ObservedObjects*: the set of entities that the sensor is designed to observe
- *JMRMIS::SensorKind*: an indicator as to whether the sensor is active or passive
- *Period*: the rate at which a sensor operates to collect sensor information

We use the *DCFM* package to specify the data correlation and fusion functions. It includes a data correlation-only function, a data correlation and fusion function, and a radar track correlation function (the threat type that is not included in the standard track representation, according to the DCFM data model). Figure 24 shows the specification for the data correlation function, including the data types expected for the incoming and outgoing ports, the data flow from source tracks to the correlated track (used in analysis of latency in the end-to-end flow), and a set of project-specific and standard properties. The data correlation and fusion function has a similar specification. We also included contractor-specific specifications of the DCFM function, highlighting differences from the specification based on the DCFM data model.

```
system DataCorrelation
features
  SourceTracks: in data port TrackTypes::DCFMSourceTrackSet;
  OwnAircraftPosition: in data port MissionSystemDataTypes::Position;
  CorrelatedTracks: out data port TrackTypes::CorrelatedThreatTrackSet;
flows
  ThreatCorrelation: flow path SourceTracks -> CorrelatedTracks;
```


Distribution Statement A: Approved for Public Release; Distribution is Unlimited

```
properties
  JMRMIS::MaxTimeStampVariation => 100 ms;
  JMRMIS::FrameOfReference => WGS84 applies to SourceTracks;
  JMRMIS::FrameOfReference => OwnAircraft applies to CorrelatedTracks;
  Transmission_Type => Pull applies to SourceTracks;
  Transmission_Type => Push applies to CorrelatedTracks;
  JMRMIS::ObservationRadius => 25 km applies to CorrelatedTracks;
  ACVIP::InputInterval => 100 ms applies to SourceTracks;
  ACVIP::OutputInterval => 1 sec applies to CorrelatedTracks;
  Latency => 1 sec .. 1 sec applies to ThreatCorrelation;
end DataCorrelation;
```

Figure 24: Data Correlation Specification

The *ASSAAssessment* package contains a specification of the situation assessment function, and the *SAAwarenessAnnunciation* package contains a specification of the situation awareness display and the situation awareness annunciation device for the aircrew. The display specification lists different data items sent to the display as separate ports. For use in the reference architecture specification, we separately defined a configurable specification of the *ASSAMFDDisplay* in the *ASSADisplayAnnunciation* package.

3.3.2 The Concept of Tracks as Observation Representation

The package *TrackTypes* contains the specification for the concept of *Track*. The requirements documents and the DCFM data model identify source tracks and correlated tracks. We introduced the additional track-related concepts of track set, track sequence, and track history, which are valuable to the specification in an AADL model.

We defined a data type *Track* with a track ID. This type indicates an instance of the track representing the position of an observed entity at a given point in time.

We added the concepts of *TrackSet* and *TrackSetDiff*. A *TrackSet* represents a collection of tracks at a given point in time, such as the set of observations made available by an ASSA sensor or the set of tracks processed by DCFM. *TrackSetDiff* represents the difference between two track sets. According to the DCFM data model, DCFM produces as output a set of changes relative to the previous track set in terms of removing, adding, and modifying tracks in the track set. This concept allows us to specify that the track set or set difference must be communicated and processed as a single consistent abstraction.

Figure 25 illustrates the definition of *Track* and *TrackSet* in AADL. The track ID is identified as a 32-bit integer, and the size of the track set is bounded by a maximum value defined as property constant *JMRMISConstants::MaxTracksInSet*.

```
data Track
properties
  ACVIP::Description => "Track represents an observed entity at a given point in
time";
end Track;
```


Distribution Statement A: Approved for Public Release; Distribution is Unlimited

```
data implementation Track.basic
subcomponents
  trackID: data Base_Types::Integer_32;
end Track.basic;

data TrackSet
properties
  ACVIP::Description => "Set of tracks represents a collection of tracked entity at a
given point in time";
end TrackSet;

data implementation TrackSet.basic
subcomponents
  elements: data Track [JMRMISConstants::MaxTracksInSet];
end TrackSet.basic;
```

Figure 25: Track Type Definitions in AADL

TrackSequence represents a track of the same observed entity over time. All elements in the sequence are assumed to have the same track ID. We can document this assumption as a predicate in an associated requirement.

TrackHistory represents a bounded track sequence, which may be used for extrapolation or for showing a trace of a track over recent time.

We have separate data types for *SourceTrack* and *CorrelatedTrack* to reflect the DCFM data model. Our specification of *SourceTrack* includes *Position*, *Velocity*, and *SamplingTime*—the latter two are not present in the DCFM data model. We elaborated *Position* with a representation specification that includes the dimensions of a position as well as the frame of reference. The package *MissionSystemDataTypes* also contains definitions for velocity, *SphericalTerrainSA*, and other data types for data being exchanged. For *CorrelatedTrack*, we specify two variants to represent two variations of the relation to source tracks used in the correlation: a list of source track IDs and a list of actual source track (copies) as part of the correlated track. The former representation assumes that the source tracks at a given point in time remain available. The latter replicates the source track, increasing the memory footprint of ASSA.

Track sets, track sequences, and correlated tracks include a specification of the maximum size of the set, sequence, or source track references.

We added the data type *AssessedTrack* to represent a track that includes the situation assessment results with respect to awareness and alert thresholds.

We introduced the *JMRMIS::FrameOfReference* property to specify the frame of reference used in the specification of a location. The frame of reference may be the World Geodetic System 1984 (WGS84) global reference system or the own aircraft position. This allows us to extend the connection-consistency check to include a consistent frame of reference across connections.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

We introduced the *JMRMIS::MaxTimeStampVariation* property to specify the maximum acceptable variation in time stamp values within a track set. This allows us to specify conditions under which variation in time results in an unsafe condition.

The MIS System Requirements Specification document distinguishes between sensor-specific track representations and a standardized track representation. The latter is specified in the DCFM data model. The *TrackTypes* package includes data types for both of these categories.

3.3.3 The Functional ASSA Architecture with Data Representations

We use the various track data types to annotate incoming and outgoing port specifications for each functional element of the ASSA system. The package *ASSASystem::Functional* defines the functional architecture of the ASSA system. The interactions shown in Figure 26 are defined by port connections.

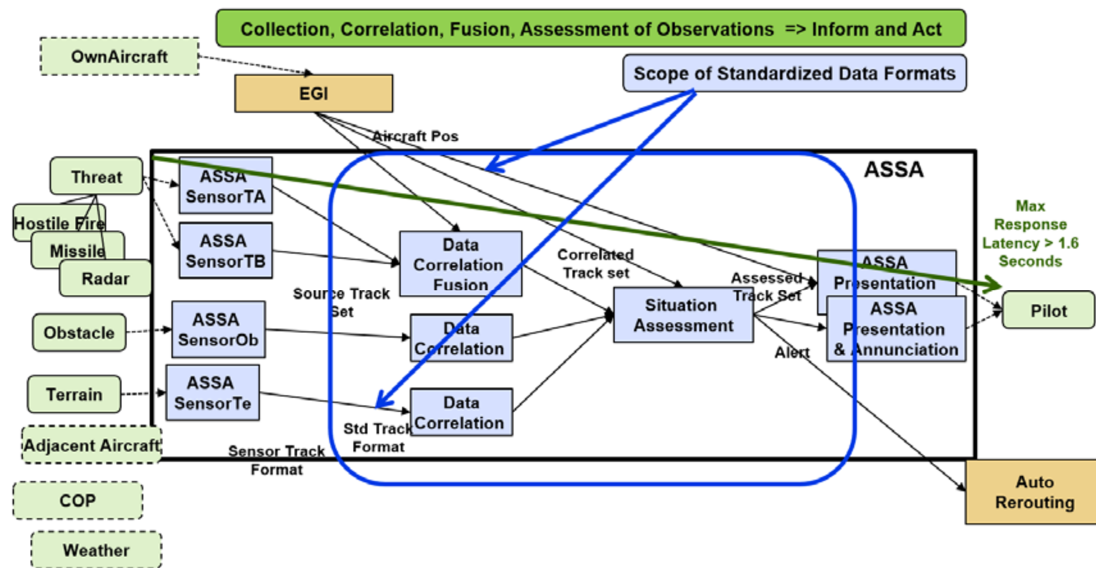

Figure 26: Scope of Standard Track Representation

The AADL compiler will check for various forms of type inconsistencies. Type inconsistencies can occur between the data types of outgoing and incoming ports on both ends of a connection. An elaborated form of consistency check includes comparison of base types, measurement units, and range of acceptable values. In our case, the checker identifies a mismatch between sensor-specific and standardized track representations (see Figure 27). They are identified at the boundary of the standardized track representation scope and are resolved by applying the ASSA data-conversion service. Such type checking also identifies potential inconsistencies in hardware connections, such as whether a system is connected to the correct type and variant of a network.

Errors (4 items)
'SourceTracks' and 'Sourcetracks' have incompatible classifiers.
'SourceTracks' and 'Sourcetracks' have incompatible classifiers.
'SourceTracks' and 'Sourcetracks1' have incompatible classifiers.
'SourceTracks' and 'Sourcetracks2' have incompatible classifiers.

Figure 27: Interaction Inconsistencies in the ASSA Functional Architecture

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

3.4 Identifying the System Boundary

The final step of this phase is the identification of the system boundary for DCFM and MIS. For DCFM, we determined the functions that must be packaged into the DCFM and which types of observations it will handle. The purple line in Figure 28 indicates the potential scope of DCFM, that is, whether it performs correlation and fusion of threats only or also terrain, obstacles, and even data from non-situational awareness sensors such as aircraft position. It also shows that DCFM receives aircraft position, which suggests that DCFM changes the frame of reference for track data from global to aircraft relative. Finally, it indicates that DCFM may perform situation assessment to determine whether a critical proximity threshold has been crossed.

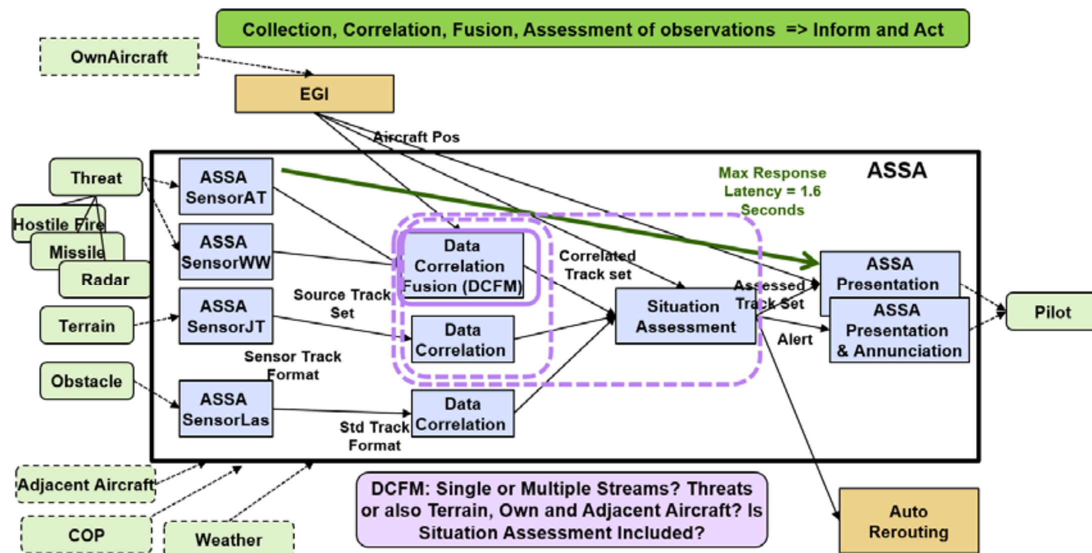

Figure 28: System Boundary of DCFM

Note that the MIS Stakeholder Requirements document discusses threats, obstacles, and terrain, while the MIS SSS and the DCFM data model mention only threats.

3.5 System Boundary and Roles of MIS

MIS provides three services for the ASSA system:

1. a data-conversion service as part of an infrastructure layer below ASSA
2. a data-storage service as part of an infrastructure layer below ASSA
3. a supervisory monitoring and control service (safety system) that oversees the nominal ASSA system operation in a supervisory layer above ASSA

The relationship between the ASSA application functionality and the MIS services is illustrated in Figure 29 in terms of a layered architecture. The SA data-conversion and data-storage services can effectively be viewed as protocols to support the flow of track information through the ASSA system. The SA data-conversion service is employed where there is a mismatch in track data representation, as identified in Section 3.3.3 and shown in Figure 29 by arrows to the appropriate connections from the SA data-conversion service. The SA data-storage service accepts output

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

from various ASSA subsystems and then makes it available to the same and other ASSA subsystems. In Figure 29, arrows from the SA data-storage service identify potential candidates of data to be handled by this service.

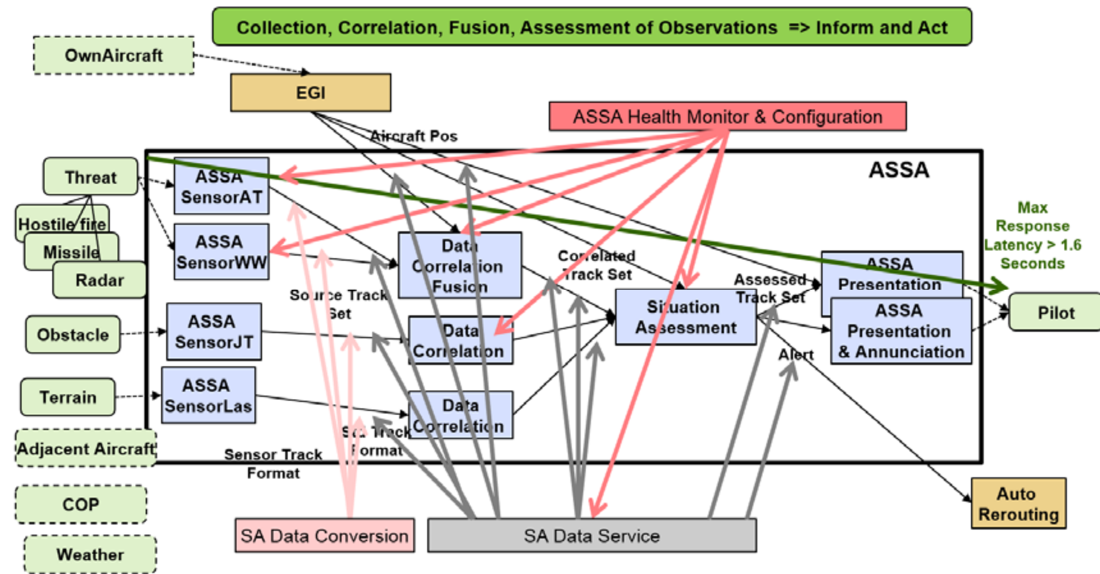

Figure 29: Identification of MIS Services as Layered Architecture

The provided documents contained ambiguous information as to the complete set of data to be handled. For example, it is unclear whether the SA data-storage service will provide own aircraft position only to DCFM or also to situation assessment and to the Multi-Function Display (MFD) for the pilot or whether the latter should come directly from the EGI source. Similarly, it is unclear whether the SA data-storage service should manage assessment results as well as other situation assessment data, such as adjacent aircraft, weather, and friendly, enemy, neutral and noncombatant (FENN) entities.

In addition, it is unclear whether the SA data-storage service is to store or even generate alerts. According to the MIS SSS, “the MIS System shall determine whether a new threat alert should be created,” and “the MIS System shall periodically evaluate alerts according to a periodic schedule defined by system configuration data to determine whether they should be deactivated.”

Capturing requirements information in an AADL model helps identify such ambiguities quickly. Figure 30 shows the AADL model equivalent to the graphical depiction in Figure 29. We defined an *ASSASystem* implementation that represents a basic configuration that gets extended with additional sensors and DCFM services as appropriate. We defined instances of the SA data-conversion and data-storage services as abstract components. We recorded the association of these services with different connections in the model by the property *ACVIP::Service_Binding*. For some connections, we identified the need for both data conversion and data storage, while for others we identified data-storage only.

```
system implementation ASSASystem.Common
subcomponents
  APR39D : device ASSASensors::APR39D;
  WW : device ASSASensors::WeaponsWatch;
```


Distribution Statement A: Approved for Public Release; Distribution is Unlimited

```
  RadarDCM : system DCFM::RadarTrackCorrelation;
  SAAssessment : system ASSAAssessment::SituationAssessment;
  AirCrewDisplay : device SAAwarenessAnnunciation::SituationAwarenessDisplay;
  AircrewAnnunciation : device SAAwarenessAnnunciation::SAAnnunciationDevice;
  ASSAHealthMonitor : abstract;
  ASSADataService : abstract ASSADataService;
  ASSADataConversion : abstract ASSADataConversion;
connections
  WeatherInfo : port Weather -> AirCrewDisplay.WeatherInformation;
  RadarTracks : port APR39D.SourceTracks -> RadarDCM.SourceTracks;
  CorrelatedRadars : port RadarDCM.CorrelatedTracks -> SAAssessment.RadarTracks;
  CorrelatedRadarTracks : port RadarDCM.CorrelatedTracks -> AirCrewDisplay.RadarInfor-
mation;
  AssessedTracks : port SAAssessment.AssessedTracks -> AirCrewDisplay.AssessedInfor-
mation;
  SAAlerts : port SAAssessment.Alerts -> AircrewAnnunciation.Alerts;
  AircrewVisuals : feature group AirCrewDisplay.AircrewSAInformation -> ASSAAirCrew-
Presentation;
  MyPositionAssessment : port OwnAircraftPosition -> SAAssessment.OwnAircraftPosition;
  MyPositionDisplay : port OwnAircraftPosition -> AirCrewDisplay.OwnAircraftPosition;
flows
  RadarAlert : end to end flow APR39D.RadarObserved -> RadarTracks ->
    RadarDCM.RadarCorrelation -> CorrelatedRadars ->
    SAAssessment.RadarAlerts -> SAAlerts -> AircrewAnnunciation.SoundAlerts;
  RadarObservation : end to end flow APR39D.RadarObserved -> RadarTracks ->
    RadarDCM.RadarCorrelation -> CorrelatedRadars ->
    SAAssessment.RadarAssessment -> AssessedTracks ->
    AirCrewDisplay.AssessedThreatInfo;
properties
  Latency => 1650 ms .. 1650 ms applies to RadarObservation, RadarAlert;
  ACVIP::Supervise => (reference (APR39D), reference (WW))
         applies to ASSAHealthMonitor;
  ACVIP::Service_Binding => (reference (ASSADataService))
         applies to CorrelatedRadarTracks, CorrelatedRadars, SAAlerts,
        MyPositionAssessment,MyPositionDisplay, WeatherInfo;
  ACVIP::Service_Binding => (reference (ASSADataService),
          reference (ASSADataConversion))
         applies to RadarTracks;
end ASSASystem.Common;
```

Figure 30: ASSA System Information Flows and the Data-Conversion and Data-Storage Services

The data types on the ports involved in various connections identify whether they expect individual tracks, the most recent track set, or a track history. This lets us determine the types of data requests that the SA data-storage service is expected to handle.

The SA health monitor is responsible for determining whether all elements of the ASSA system are operational. In other words, it acts as the safety system for ASSA to ensure that the aircrew can safely operate the aircraft from an aircraft survivability perspective. Potential issues with its specification will be discussed in the context of the ASSA system safety analysis [Feiler 2015c].

3.6 ASSA Functional Architecture Performance

At this stage, we can consider two types of performance-related quality attributes:

1. response time of the ASSA system for informing the aircrew about a threat, obstacle, or terrain
2. expected maximum data volume to be processed

3.6.1 ASSA System Response Time Analysis

As we investigated response time requirements for the ASSA system, we defined the end-to-end flow from the time that a sensor detects a threat to the time that the pilot sees the threat on the display. We annotated this definition with the appropriate *Latency* property value. This response time is of interest to stakeholders in the ASSA system, such as the pilot.

MIS documentation mentions an expected latency of 1,600 milliseconds (ms) for threats. The requirement statement does not distinguish between different types of threats. Also, there are no requirement statements about latency for obstacles or terrain. In the AADL model, we can add flows with the appropriate latency values as necessary.

We annotated each of the functional subsystems with various types of performance-related properties. Where appropriate we defined the *Period* at which a functional unit is intended to operate. The requirement documents indicate that ASSA and MIS services may operate at rates between 100 ms and 1 s. For the purpose of response time analysis, we assume a best case of 100 ms.

We also defined flow specifications for each component, including flow sources when data starts within a component, flow paths from a component input to its output, and flow sinks when data flow ends within a component. For each of those flows, we specified an expected latency value. In addition, we specified latency contributions by the SA data-conversion and data-storage services.

For the SA data-storage service, documentation indicates that clients request data from it. This "pull" protocol adds one frame of communication delay, in the best case of 100 ms, if the SA data-storage service and the client reside in different partitions.

Within the *ASSASystem* implementation declaration, we included end-to-end flow specifications for radar and for hostile fire threats. For each we defined an end-to-end flow from the sensor to present the correlated track on the display without going through situation assessment (*RadarObservation* and *HostileFireObservation*), an end-to-end flow from the sensor to provide a warning as result of situation assessment (*RadarAssessment* and *HostileFireAssessment*), and an end-to-end flow from the sensor to the annunciation device to provide an alert as result of situation assessment (*RadarAlert* and *HostileFireAlert*).

After we created an instance model for the ASSA system and ran the end-to-end latency analysis, we got a first set of results that accounted for latency contributions by all the functional units as

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

well as the SA data-conversion and SA data-storage services. The AADL Wiki provides details on how the latency analysis works and how to interpret the results [SEI 2015].

Figure 31 shows the analysis results for *RadarObservation*. The maximum latency is less than the expected maximum latency, and the same is true for the minimum latency. In addition, the analysis compares the expected latency jitter against the calculated latency jitter, that is, the difference between the minimum and maximum latency. In this case, the calculated jitter is less than the expected jitter.

Latency analysis for end-to-end flow 'RadarObservation' of system 'ASSASystem.DCMConceptual' with latency preference stettings AS-PE-ET-EQ							
Contributor	**Min Speci**	**Min Value**	**Min Method**	**Max Speci**	**Max Value**	**Max Method**	**Comments**
device APR39D		0.0ms	first sampling		0.0ms	first sampling	Initial 100.0ms sampling latency not added
device APR39D	25.0ms	25.0ms	specified	50.0ms	50.0ms	specified	
(system ASSADataService)	100.0ms	100.0ms	specified	100.0ms	100.0ms	specified	Using specified bus latency
(system ASSADataConversion)	3.0ms	3.0ms	specified	5.0ms	5.0ms	specified	Using specified bus latency
Connection		103.0ms	no latency		105.0ms	no latency	Adding latency subtotal from protocols and bus - shown with ()
system RadarDCM		0.0ms	sampling		0.0ms	sampling	Best case 0 ms, worst case 0.0ms (period) sampling delay
system RadarDCM	1000.0ms	1000.0ms	specified	1000.0ms	1000.0ms	specified	
(system ASSADataService)	100.0ms	100.0ms	specified	100.0ms	100.0ms	specified	Using specified bus latency
Connection		100.0ms	no latency		100.0ms	no latency	Adding latency subtotal from protocols and bus - shown with ()
device AirCrewDisplay		0.0ms	sampling		100.0ms	sampling	Best case 0 ms, worst case 100.0ms (period) sampling delay
device AirCrewDisplay	1.0ms	1.0ms	specified	1.0ms	1.0ms	specified	
Latency Total	1129.0ms	1229.0ms		1156.0ms	1356.0ms		
End to End Latency		1400.0ms			1600.0ms		
End to end Latency Summary							
SUCCESS	Actual end-to-end flow latency jitter for RadarObservation is within specified end to end latency jitter and minimum reposne time is better						

Figure 31: Response Time for Observed Enemy Radar Track

Figure 32 shows the analysis results for *RadarAssessment* when the processing path includes situation assessment processing. In this case, the numbers are slightly higher. Both the maximum and minimum calculated latency are less than the expected values. However, the calculated latency jitter is larger than the expected latency jitter; thus, it produces an error message.

Latency analysis for end-to-end flow 'RadarAssessment' of system 'ASSASystem.DCMConceptual' with latency preference stettings AS-PE-ET-EQ							
Contributor	**Min Speci**	**Min Value**	**Min Method**	**Max Speci**	**Max Value**	**Max Method**	**Comments**
device APR39D		0.0ms	first sampling		0.0ms	first sampling	Initial 100.0ms sampling latency not added
device APR39D	25.0ms	25.0ms	specified	50.0ms	50.0ms	specified	
(system ASSADataService)	100.0ms	100.0ms	specified	100.0ms	100.0ms	specified	Using specified bus latency
(system ASSADataConversion)	3.0ms	3.0ms	specified	5.0ms	5.0ms	specified	Using specified bus latency
Connection		103.0ms	no latency		105.0ms	no latency	Adding latency subtotal from protocols and bus - shown with ()
system RadarDCM		0.0ms	sampling		0.0ms	sampling	Best case 0 ms, worst case 0.0ms (period) sampling delay
system RadarDCM	1000.0ms	1000.0ms	specified	1000.0ms	1000.0ms	specified	
(system ASSADataService)	100.0ms	100.0ms	specified	100.0ms	100.0ms	specified	Using specified bus latency
Connection		100.0ms	no latency		100.0ms	no latency	Adding latency subtotal from protocols and bus - shown with ()
system SAAssessment		0.0ms	sampling		100.0ms	sampling	Best case 0 ms, worst case 100.0ms (period) sampling delay
system SAAssessment	10.0ms	10.0ms	specified	10.0ms	10.0ms	specified	
Connection		0.0ms	no latency		0.0ms	no latency	
device AirCrewDisplay		0.0ms	sampling		100.0ms	sampling	Best case 0 ms, worst case 100.0ms (period) sampling delay
device AirCrewDisplay	1.0ms	1.0ms	specified	1.0ms	1.0ms	specified	
Latency Total	1139.0ms	1239.0ms		1166.0ms	1466.0ms		
End to End Latency		1400.0ms			1600.0ms		
End to end Latency Summary							
ERROR	Jitter of actual latency total 227.0 ms exceeds expected end to end latency jitter 200.0ms with minimum actual latency total less then expected minimum						

Figure 32: Response Time for Assessed Enemy Radar Track

In Section 4.2, we revisit the latency analysis on the ASSA system as elaborated into a design architecture.

3.6.2 Track Data Volume

The volume of track data to be processed is essential for determining the memory footprint of the SA data-storage service as well as that of functional units. In addition, the volume affects transmission time between functional units and processing time by the units.

We introduced two properties—*ACVIP::OutputInterval* and *ACVIP::InputInterval*—that allow the modeler to specify the rate at which input is expected and output is intended to be sent. As we

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

do this for each functional unit, a consistency checker can ensure that the outgoing rate is consistent with the incoming rate for each connection. The data volume is also affected by the size of each track set being communicated. To account for this variability, we specified maximum track set sizes using the *Data_Model::Dimension* property. For the property value, we used a property constant. The property constants themselves are defined in the property set *JMRMISConstants*. The constants allow us to change the sizes of various types of track sets in a single place without searching through the model for each occurrence.

4 ASSA System Design Architecture

In this section, we summarize the ASSA system design architecture, in which the SA data-conversion and SA data-storage services are explicitly included. We also provide a specification of interaction protocols used in communicating SA data between subsystems. We do so by defining the protocol as a virtual bus with the appropriate latency contribution specified as the *Latency* property. We also model each protocol implementation as an AADL model and analyze it to verify that the latency value used in the protocol specification is correct.

4.1 ASSA Design Architecture

The ASSA system design architecture is specified in the package *ASSASystem::Design*. We created two variants of the design architecture:

1. *ASSASystem.MISDataConversionArchitecture:* In this architecture, the SA data-conversion service is inserted into the interactions between ASSA components, while the SA data storage is represented as a service in a layer below. In this case, the information flow is still apparent in the model.
2. *ASSASystem.MISDataServiceArchitecture:* In this architecture, the SA data-storage service is also inserted into the interactions between ASSA components. In this case, all interactions between ASSA components become interactions with the SA data-storage service. The result is a model in which the information flow between ASSA components becomes implicit in the storage and retrieval order of data to and from the SA data-storage service (see Figure 33).

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

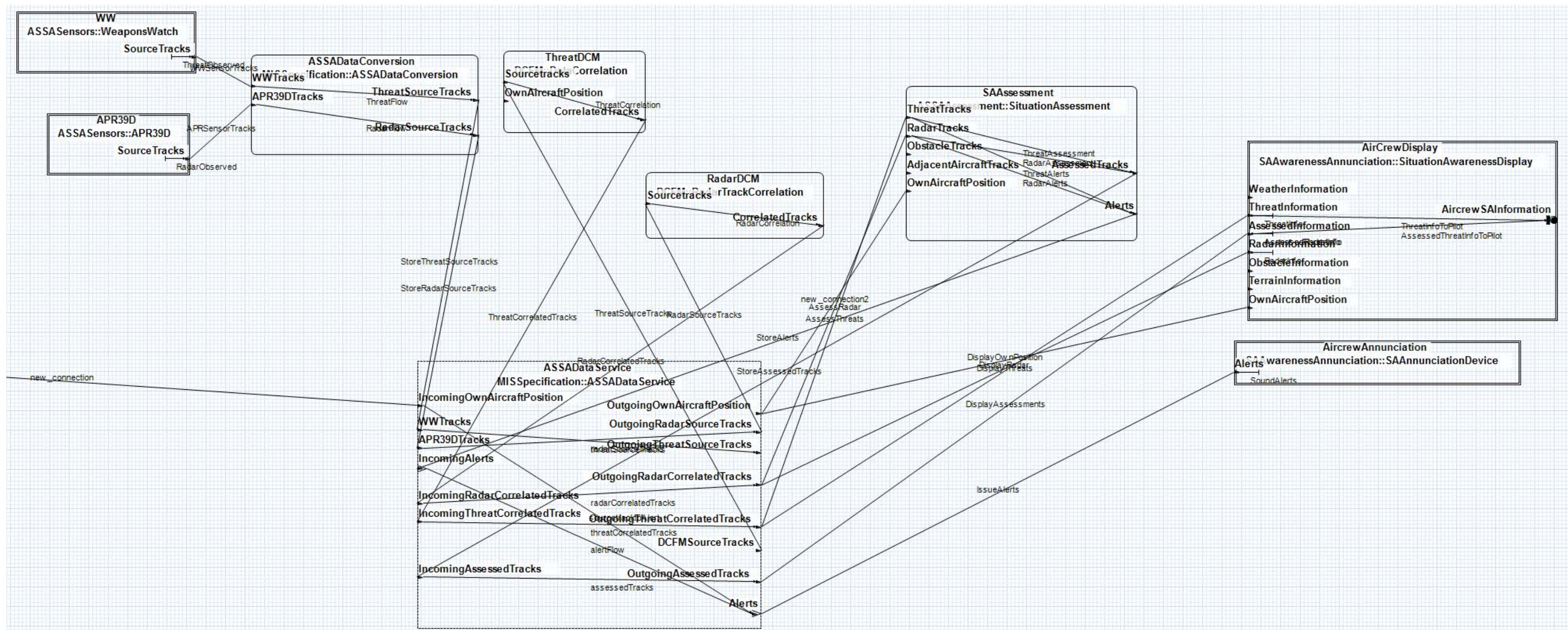

Figure 33: ASSA System Design Architecture with SA Data-Conversion and Data-Storage Services

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

4.2 Performance Analysis of the ASSA System Design Architecture

In this section, we revisit response time requirements and perform appropriate analysis on this more detailed architecture. We also represent the interactions between the ASSA application components as protocols and model the implementation of each protocol in AADL in the context of an ARINC 653–compliant partitioned architecture to determine the latency contribution of each interaction.

4.2.1 Response Time Requirement Revisited

The MIS SSS document states that "The MIS System shall publish Correlated Tracks to Client Systems in less than 1,600 ms measured from the pop-up threat arrival time from one or more AST Systems." When we interpret this statement in the context of the ASSA design architecture, we realize that the response time is the flow from the time that track data arrives from one of the ASSA sensors to the time that data is sent to the display (see Figure 28). This is different from the end-to-end flow specification in the previous section, which reflected the notion of response time useful to stakeholders. This distinction allows us to clarify that the 1,600-ms response time was not intended to reflect the response time from threat appearance to observation by the aircrew but instead that the response time will actually be larger and thus present a greater risk.

4.2.2 Interaction Protocols

From the requirements documents, we gathered that the communication from the SA sensors to the SA data service is a "push" architecture; that is, the data is transferred at the rate the sensors produce it. The DCFM and the aircrew display request data to be sent at the rate of the receiver, which is a "pull" architecture. AADL has a *Transmission_Type* property that allows modelers to indicate which connections assume a push or a pull.[6]

We also know that different ASSA system functions reside in different partitions: data conversion, DCFM, SA data storage, and display. At this stage of modeling, we can reflect cross-partition communication and the resulting latency contribution in a protocol abstraction modeling the details of a particular partition configuration. Later, we can elaborate the architecture, map each functional unit into the appropriate partitions, and revisit the latency analysis to ensure that the abstractions we introduced here are consistent.

The package *MISProtocols* contains several virtual bus specifications, each reflecting a different protocol. Different connections of the ASSA system design architecture include a specification of the type of protocol that we expect to be used, which is expressed by the *Required_Virtual_Bus_Class* property. The end-to-end latency analysis will take this property into account in order to include latency contributions by the respective protocol. The protocols are:

- *SensorPushProtocol:* communication from the SA sensor to the SA data-conversion function (also referred to as "sensor manager" in some documents). Its latency contribution can vary between zero and a maximum of a major partition frame, depending on the alignment of the

---

[6] Push/pull is different from publish/subscribe. Publish/subscribe allows a system to dynamically establish connections without knowing the other party. In push/pull, the sender or receiver determines the transmission rate over an established connection.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

recipient partition windows with the sensor dispatch schedule. Note that transfer latency by a physical bus will be represented separately by an AADL bus component.

- *PullProtocol:* a single request/reply cycle across a partition boundary. Its latency is specified as a minimum and maximum of 100 ms—the smallest partition rate.
- *PullDCFMInputDataSetProtocol:* an abstraction of the DCFM interaction protocol specified as a sequence diagram in the original documentation. The diagram specifies that DCFM interacts with SA data storage through three sequential request/reply actions. We specify its latency as a minimum and maximum of 300 ms.
- *SADataServiceProtocol:* an abstraction of the SA data service as a communication protocol. We use it in the *MISDataConversionArchitecture* variant of the ASSA system design architecture and verify it in the *MISDataServiceArchitecture* variant.

The package also contains models of the implementation of these protocols running on a partitioned processor system. We use these implementation models to determine the latency contribution by the protocol, that is, to verify the latency value used in the virtual bus specification of the protocol. The implementation model has a sending thread and a receiving thread. Each thread has a set of ports that represent the request/reply interactions. The sender/receiver interaction is represented by an end-to-end flow, which is shown graphically in Figure 34.

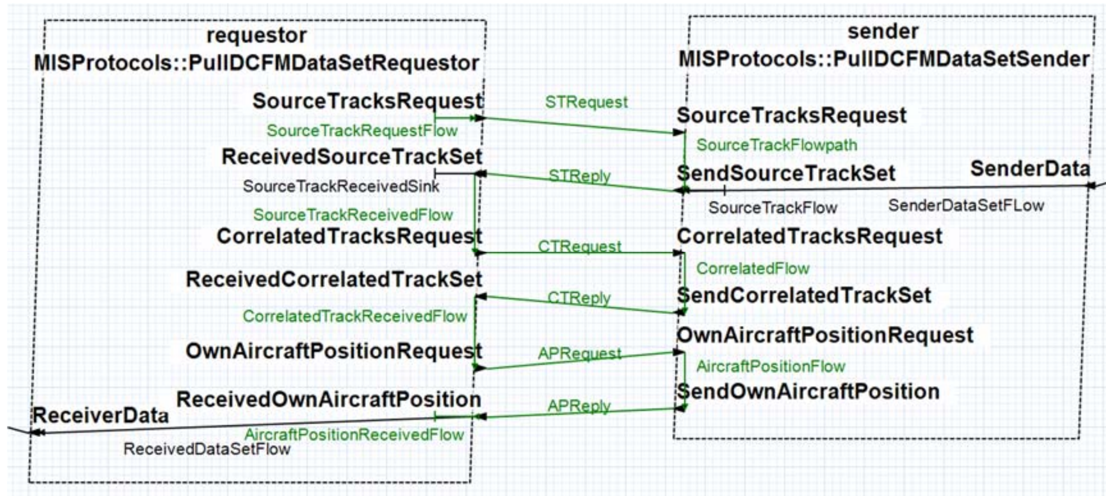

Figure 34: Three-Step Pull Protocol

We modeled the protocol implementation in two ways:

1. *Immediate and delayed connections:* In this case, we use the semantics of the AADL immediate connection (communication within the same frame) and delayed connection (communication to the next frame) to reflect the cross-partition delay in one of the two directions through the connection semantics. Depending on whether the sender or the receiver has the earlier partition window, the frame boundary is crossed on reply or on request (delayed connection).
2. *Explicit partition binding:* In this case, we introduce a processor with two partitions and bind the sender and the receiver to one partition each. Then we validate that one direction is frame-delayed based on actual partition schedules. We can switch the binding of the sender

and receiver to demonstrate that the result is independent of the partition order in the schedule. We can also explore whether placing partitions on different processors creates additional latency overhead.

In the AADL model, we defined the protocol once and then configured it as the two implementation variants, as shown in Figure 35. Each configuration is instantiated and analyzed for end-to-end latency. The next section discusses the results of this analysis.

```
abstract implementation PullInputDataset.CrossPartitionTasks
    extends PullInputDataset.Common
properties
 Timing => immediate applies to STRequest, CTRequest, APRequest;
 Timing => delayed applies to STReply, CTReply, APReply;
end PullInputDataset.CrossPartitionTasks;

abstract implementation PullInputDataset.PT extends PullInputDataset.Common
subcomponents
  pform : processor ASSAHardware::GPU.TwoPartition;
properties
  Actual_Processor_Binding => (reference(pform.part1)) applies to requestor;
  Actual_Processor_Binding => (reference(pform.part2)) applies to sender;
end PullInputDataset.CrossPartitionTasks;
```

Figure 35: Two Implementation Configurations of the Three-Step Protocol

4.2.3 Response Time Analysis

In this example, we demonstrate compositional response time analysis in two steps. First, we perform end-to-end latency analysis for each protocol implementation to verify that the computed latency of the implementation corresponds to the specified latency values in the virtual bus abstraction. Second, we verify the end-to-end latency for the ASSA system based on the required binding specifications to the virtual buses representing the protocol abstraction.

In the case of the *PullDCFMInputDataSetProtocol*, we get a latency of 600 ms. This is due to the default preference setting for latency analysis. By default, cross-partition data transfer is assumed to occur at the end of a major frame. There is a delay of 100 ms for each of the three requests and replies. If we change the setting in the OSATE Preferences for latency analysis to *Partition End* output policy so that output is flushed at partition end rather than frame end, we get 100 ms for each request/reply pair of the protocol. We reflect these results in the latency specification of the virtual bus that represents the respective protocol.

When we instantiate the design architecture of ASSA and perform latency analysis, we include the latency contributions of these protocols. The result is shown in Figure 36. It indicates that for the synchronous system case, the response time exceeds the required 1,600 ms.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Latency analysis for end-to-end flow 'ThreatInfo' of system 'ASSASystem.MISDataConversionArchitecture' with latency preference stettings SS-MF-							
Contributor	**Min Specif**	**Min Value**	**Min Metho**	**Max Speci**	**Max Value**	**Max Metho**	**Comments**
device WW		0.0ms	first sampli		0.0ms	first sampli	Initial 100.0ms sampling latency not added
device WW	20.0ms	20.0ms	specified	30.0ms	30.0ms	specified	
Connection		0.0ms	no latency		0.0ms	no latency	
abstract ASSADataCor		0.0ms	sampling		0.0ms	sampling	Min: Roun(Assume sy Max: Round up to sampling period 0.0ms
abstract ASSADataCor	3.0ms	3.0ms	specified	5.0ms	5.0ms	specified	
(Protocol MISProtocols	300.0ms	300.0ms	specified	300.0ms	300.0ms	specified	Using specified bus latency
(Protocol MISProtocols	100.0ms	100.0ms	specified	100.0ms	100.0ms	specified	Using specified bus latency
Connection		400.0ms	no latency		400.0ms	no latency	Adding latency subtotal from protocols and bus - shown with ()
system ThreatDCM		0.0ms	sampling		0.0ms	sampling	Min: Roun(Assume sy Max: Round up to sampling period 0.0ms
system ThreatDCM	1000.0ms	1000.0ms	specified	1000.0ms	1000.0ms	specified	
(Protocol MISProtocols	100.0ms	100.0ms	specified	100.0ms	100.0ms	specified	Using specified bus latency
Connection		100.0ms	no latency		100.0ms	no latency	Adding latency subtotal from protocols and bus - shown with ()
system SAAssessment		0.0ms	sampling		0.0ms	sampling	Min: Roun(Assume sy Max: Round up to sampling period 100.0ms
system SAAssessment	10.0ms	10.0ms	specified	10.0ms	10.0ms	specified	
(Protocol MISProtocols	100.0ms	100.0ms	specified	100.0ms	100.0ms	specified	Using specified bus latency
Connection		100.0ms	no latency		100.0ms	no latency	Adding latency subtotal from protocols and bus - shown with ()
device AirCrewDisplay		90.0ms	sampling		90.0ms	sampling	Min: Roun(Assume sy Max: Round up to sampling period 100.0ms
device AirCrewDisplay	1.0ms	1.0ms	specified	1.0ms	1.0ms	specified	
Latency Total	1334.0ms	1724.0ms		1346.0ms	1736.0ms		
End to End Latency		1400.0ms			1600.0ms		
End to end Latency Summary							
WARNING	Minimum specified flow latency total 1334.0ms less then expected minimum end to end latency 1400.0ms (better response time)						
ERROR	Minimum actual latency total 1724.0ms exceeds expected maximum end to end latency 1600.0ms						
ERROR	Maximum actual latency total 1736.0ms exceeds expected maximum end to end latency 1600.0ms						

Figure 36: Response Time Analysis Results with Latency Contributions for the Cross-Partition Pull Protocol

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

5 Maintaining Requirement Specifications in ALRS

The AADL specification of a system such as *ASSA* or *DCFM*—expressed as an annotated system type, abstract type, or device type—acts as a requirements specification in two ways:

1. The customer or acquirer has developed a specification. The supplier response, also expressed as an AADL specification, is compared against the original for compliance to determine whether it meets the customer's interest.
2. A specification represents a contract that a system implementation must meet. In this case, AADL models of the system architecture design, detailed design models in other notations, and code must be verified against this specification.

In this section, we introduce the ReqSpec notation, which allows us to explicitly identify different parts of this system specification as a set of traceable requirements whose verification and satisfaction can be demonstrated by a set of verification actions. A more complete description of ReqSpec will appear in a forthcoming report.[7]

5.1 The ReqSpec Notation

In this case study, we use ReqSpec, the requirements specification subset of the RDAL draft standard metamodel in textual form. The notation has its roots in goal-oriented requirements engineering, which distinguishes between stakeholder requirements, referred to as goals, and system requirements, referred to as requirements. Goals express stakeholder intent and may conflict with each other, while system requirements represent a contract that the system implementation must meet.

The notation accommodates several capabilities:

- Import of existing stakeholder and system requirements documents, such as from DOORS, allows users to examine and reference them without the respective external tool.
- Definition of placeholders for other external documents, such as the MIS BAA Supplement, allow for references into these documents by other RDAL elements.
- Goals and requirements can be associated with an architecture model expressed in AADL. These goals and requirements may have been imported from existing documents, or they may have been specified separately in the context of an architecture model. In the latter case, ReqSpec maintains traceability to existing requirements documents.
- ReqSpec facilitates an explicit record of goal and requirement refinement, decomposition, and evolution. This record may identify conflicts between goals.
- Association of verification actions with requirements can specify how they are to be verified and satisfied.

---

[7] Feiler, P. and Delange, J. *A Requirement Specification Language for AADL.* CMU/SEI-2015-SR-034. Software Engineering Institute, Carnegie Mellon University. Forthcoming.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

5.2 Goal and Requirement Specifications

RDAL goal and requirement specifications are associated with AADL component types, component implementations, and elements within them. A goal or requirement specification has the following elements. All but the name are optional.

- *name:* unique identifier with respect to other goals (or requirements) for the same component
- *title:* a short descriptor of the goal or requirement
- *for:* reference to a model element within a component, such as a port or end-to-end flow
- *category*: indicator of a goal or requirement category (e.g., assumption, guarantee, safety, performance). Categories are user definable.
- *description:* a long descriptor of the goal or requirement
- *a set of variables:* used to parameterize goal and requirement specifications. Many of the changes to a goal or requirement are in a value used in the goal or requirement specification. Variables allow users to define a requirement value once and reference it in the description, predicates, and verification activities of verification plans expressed in a Verify notation[8].
- *rationale:* rationale for the goal or requirement
- *refines:* reference to another goal (requirement) of the same component that this goal (requirement) refines. This represents the refinement of goals (requirements) for a specific system into more detailed requirement specifications for the same component.
- *conflicts with (goal only):* list of other goals that this goal may conflict with
- *evolves:* reference to a goal (requirement) of the same component that another goal (requirement) evolves to. This provides a record of a goal (requirement) evolving into another specification over time. *Dropped* is used to indicate that the original requirement of the evolved requirement is not relevant any more.
- *stakeholder (goal only):* list of stakeholder references for a given goal
- *see:* reference to a model element or property in the model that represents the requirement in the model
- *see document requirement:* reference to a stakeholder requirement in an existing document
- *see document:* reference to an external document and element within expressed as a Uniform Reference Identifier (URI). It records the fact that a stakeholder requirement is found in a document other than an imported requirement document.
- *issues:* list of text strings expressing issues with respect to the goal or requirement

In the ReqSpec syntax, the elements within the square brackets can be declared in any order. The syntax of a goal declaration is as follows:

```
Goal ::=
goal Name ( :  Title )?
  ( for TargetElement )?
```

[8] The Verify notation is part of a set of notations developed under the Incremental Lifecycle Assurance project and will be published in early 2016.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

```
[
  ( category ( <ReqCategory> )+ )?
  ( description )?
  ( ConstantVariable )*
  ( rationale String )?
  ( refines ( <Goal> )+ )?
  ( conflicts with ( <Goal>
)+)?
  ( evolves ( <Goal> )+)?
  ( dropped )?
  ( stakeholder ( <Stakeholder> )+ )?
  ( see document requirement ( <Requirement> )+)?
  ( see document ( DocReference )+ )?
  ( issues (String)+ )?
  ( ChangeUncertainty )?
]

Title ::= String
TargetClassifier ::= <AADL Component Classifier>
TargetElement ::= <ModelElement>
DocReference ::= URI to an element in an external document
```

The following elements are for requirement specifications only:

- *predicate:* a formalized specification of the condition that must be met to indicate that the requirement is satisfied. The predicate may refer to variables defined as part of this requirement or the enclosing requirement specification container.
- *mitigates:* reference to one or more hazards that the requirement addresses. A hazard is represented by an error propagation in an error model EMV2 subclause for a component.
- *decomposes:* reference to a goal (requirement) of an enclosing component. This represents the decomposition of system goals (requirements) into goals (requirements) of subsystems.
- *development stakeholder:* reference to a stakeholder from the development team, such as a security engineer or a tester. During architecture design, design choices may lead to new requirements, whose stakeholder is the developer making the choice.
- *see goal:* reference to a (stakeholder) goal that the (system) requirement is related to

The syntax of a requirement specification declaration is as follows:

```
Requirement ::=
requirement Name ( : Title )?
  ( for TargetElement )?
[
  ( category ( <ReqCategory> )+ )?
  ( description )?
  ( Variable )*
```


Distribution Statement A: Approved for Public Release; Distribution is Unlimited

```
  ( Predicate )?
  ( rationale String )?
  ( mitigates ( <Hazard> )+ )?
  ( refines ( <Requirement> )+)?
  ( decomposes ( <Requirement> )+)?
  ( evolves ( <Requirement>
)+)?
  ( dropped )?
  (development stakeholder ( <Stakeholder> )+ )?
  ( see goal ( <Goal> )+)?
  ( see document goal ( <Goal> )+)?
  ( see document requirement ( <Requirement> )+)?
  ( see document ( DocReference )+ )?
  ( issues (String)+ )?
  ( ChangeUncertainty )?
]
```

5.3 ReqSpec Files

ReqSpec declarations are not embedded in the AADL through annex clauses. Instead they are placed in separate files with the extension *goals* for a set of stakeholder goals, *reqspec* for a set of system requirements, *goaldoc* for stakeholder goal documents, and *reqdoc* for system requirement documents in document section format to mirror existing text documents.

The *StakeholderGoals* construct is a container for *Goal* declarations that are associated with a specific system. The system is identified by its AADL component classifier declaration using the *for* clause. The *NestedName* can be an identifier or a sequence of identifiers separated by a <dot>, for example, *aircraft.system*. This is similar to AADL package names where *::* is used as separator.

The *ConstantVariable* is available to all goal declarations in this *StakeholderGoals* container.

```
StakeholderGoals ::=
stakeholder goals NestedName ( :  Title )?
 for TargetClassifier
[
  (description )?
  (see document ( DocReference )+ )?
  ( ConstantVariable )*
  ( Goal )+
  ( issues (String)+ )?
]
```

The *SystemRequirements* construct is a container for *Requirement* declarations. It is typically used to group together system requirements for a particular system; for components of a specified cate-

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

gory, such as a requirement for all processors to be schedulable; or for components of all categories. The system is identified by its AADL component classifier declaration using the *for* clause. Separately defined constants (*use constants*) and locally defined variables are available to all requirements within the *SystemRequirements* container.

```
SystemRequirements ::=
system requirements NestedName ( :  Title )?
 for ( TargetClassifier | ComponentCategory | all )
 ( use constants <GlobalConstants>* )?
[
  ( description  String )?
  (see document ( DocReference )+ )?
  ( Variable )*
  ( Requirement )*
  ( issues (String)+ )?
]
```

Goals and requirements can be referenced by qualified name—that is, by the *SystemRequirements* name and the *Requirement* name—separated by a <dot>. For an example, see Figure 39, which shows a reference from a requirement to a goal. In some cases, qualification is not necessary, such as when a requirement refines another requirement and both are declared in the same *SystemRequirements* container.

The *Document* construct represents existing stakeholder goals or system requirements documents that are imported into a ReqSpec representation. In these documents, goals and requirements are organized into sections. Once an existing stakeholder requirements or system requirements document has been imported into ReqSpec, users can associate its goals with an AADL model and perform traceability and consistency checks on stakeholder goals and system requirements.

A *Document* contains a set of document sections, stakeholder goals, or system requirements. A *DocumentSection* can recursively contain document sections, stakeholder goals, or system requirements. A *Document* represents a name scope for the *goal* and *requirement* declarations contained in it; a goal or requirement is referenced by the *Document* name and the *goal* or *requirement* name, separated by a <dot>. Document sections do not contribute to qualifying a name. This means that *goal* and *requirement* declarations must be unique within the *Document*.

```
Document ::=
document Name ( :  Title )?
[
  (description  String )?
  ( Goal | Requirement | DocumentSection )+
  (issues (String)+ )?
]


DocumentSection ::=
section Name ( :  Title )?
```


Distribution Statement A: Approved for Public Release; Distribution is Unlimited

```
[
  (description  String )?
  ( Goal | Requirement | DocumentSection )+
  (issues (String)+ )?
]
```

The *organization* notation allows users to define organizations and stakeholders that belong to organizations. Stakeholder names must be unique within an organization. Stakeholders are referenced by qualifying them with the organization name. Each organization is declared in a separate file with the extension *org*.

```
Organization::=
organization Name
  ( Stakeholder )+

Stakeholder ::=
stakeholder Name
[
  ( full name String )?
  ( title String )?
  ( description String )?
  ( role String )?
  ( email String )?
  ( phone String )?
]
```

The following is an example set of stakeholder declarations.

```
organization mrj
stakeholder cs
[
        full name "Claude Shannon"
        title "Lead Engineer"
        description "Mission system user"
        role "JMR representative"
]

stakeholder hl
[
        full name "Heddy Lamarr"
        title "Principal Researcher"
        description "System architect"
        email "heddylamarr@screensiren.com"
        phone "555-555-5555"
        role "Responsible for ASSA System Design"
```


Distribution Statement A: Approved for Public Release; Distribution is Unlimited

```
]
```

Requirement categories are user-definable in files with the extension *cat*. The following is a sample.

```
requirement categories
[
 safety
 security
 performance
]
```

5.4 ASSA System Goal and Requirements Specification

We used ReqSpec declarations in two ways for the ASSA system. First, we imported the content of the MIS Stakeholder Requirements document and the MIS System Requirements Specification document into the OSATE environment (*MISStakeholderRequirements.goaldoc* and *MIS-SSS.reqdoc*). Figure 37 illustrates the result of importing a stakeholder document. Users can then develop an AADL model to represent concepts, entities in the operational environment, and system components and identify them in goals with a *for* clause. Then a ReqSpec tool can identify whether requirements in a document section address a single system or refer to multiple subsystems at different architecture levels (as illustrated notionally in Figure 3).

```
document MISStakeholderRequirements [
  section ActualRequirements [
    goal SR_57 : "MIS shall operate during all aircraft operations missions and flight
profiles" [
      stakeholder mrj.hl
    ]
    goal SR_56 : "MIS shall operate during visual meteorological conditions" [
      stakeholder mrj.hl
    ]
```

Figure 37: Sample of Imported Stakeholder Requirements

Second, we created a set of ReqSpec *stakeholder goal* and *system requirement* declarations that are associated with a system represented in an AADL model. In this case, the requirements are organized around elements in the system architecture. Figure 38 shows an example of a set of *goals* specified for *ASSASensor*. The name of the stakeholder goal set mirrors the qualified name of the system classifier, but does not have to do so. Each *goal* specification has a unique name within the goal set. In our example, it includes a *title*, *description*, stakeholder reference, and list of references to the MIS Stakeholder Requirements document.

```
stakeholder goals ASSASensors.ASSASensor for ASSASensors::ASSASensor [
  goal goal1 : "Passive ASE (ASSA sensor type}"
  [ description "MIS shall support passive SA sensors (ASE)"
    stakeholder mrj.cs
    see document requirement MISStakeholderRequirements.SR_13
      MISStakeholderRequirements.SR_69 MISStakeholderRequirements.SR_15
```


Distribution Statement A: Approved for Public Release; Distribution is Unlimited

```
  ]
]
```

Figure 38: Goal Set for ASSA Sensors

Figure 39 illustrates a set of system requirements for a particular component. The component type is identified by the *for* statement. The first requirement is associated with the component. It utilizes a constant variable (*val*) to specify a requirement value that may change. The variable is referenced in the description text. The requirement also includes a cross-reference to a goal in the imported stakeholder goals document.

The second requirement is defined for an element of the component type; a particular port is identified by the *for* clause. The requirement declaration includes a constant variable for the desired observation radius. This variable is referenced in the description text as well as in the *value predicate* specification. A second variable (*compute*) is defined as a placeholder for values computed by verification methods used to verify this requirement. The *value predicate* specifies how the system will compare the desired (required) value and the value resulting from an analysis, simulation, or test run.

```
system requirements PassiveSensorReqs for ASSASensors::PassiveTerrainSensor
[
  requirement Req4 : "Passive sensor"
 [
  val EnergyLevel = 0
  description "Passive sensor radiates " EnergyLevel " energy"
  see document goal MISStakeholderRequirements.SR_27
 ]
 requirement Req1 : "Spherical terrain awareness for aircrew"
 for  TerrainSphere
 [
  description "Spherical SA of terrain within " DesiredObservationRadius " radius for 
aircrew"
  val DesiredObservationRadius = 5 nm
  compute  MeasuredDistance
  value predicate  MeasuredDistance >= DesiredObservationRadius
  see document goal MISStakeholderRequirements.SR_27
 ]
]
```

Figure 39: Example of Requirement Specification Aligned with an AADL Model

We also defined stakeholders and requirement categories. Figure 40 shows all the files in the AADL Navigator View in OSATE.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

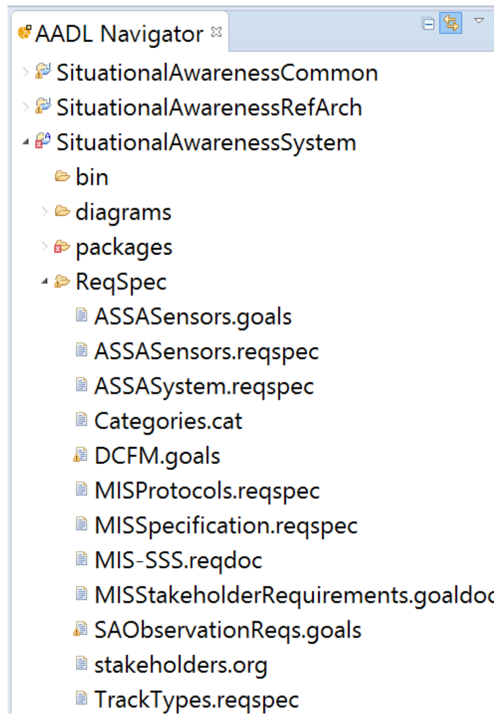

Figure 40: Project with AADL Model Packages and ReqSpec Files

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

6 Configurable Reference Architecture for Situation Assessment

This section describes how we defined a configurable reference architecture for situation assessment in AADL. We followed an approach that has been demonstrated for the JPL Mission Data System reference architecture [Feiler 2010].[9] The AADL model of this reference architecture can be found in the project *SituationalAwarenessRefArch.* This project uses some AADL packages in the project *SituationalAwarenessCommon*.

6.1 Configurable Services and Interface Specifications

We specify the ASSA system as a set of services for SA data conversion, data correlation and fusion, data storage, situation assessment, and information preparation for display on an MFD. They are specified as AADL processes to indicate each is intended to reside in a separate, protected address space. They are defined in the *ASSAServices* package.

The interface with other services is expressed as a feature group with an empty feature group type. The feature group types are specified without features in the *ASSAInterfaces* package (found in the *SituationalAwarenessCommon* project). We will refine this feature group type into one specific to a particular aircraft configuration.

Figure 41 shows *ASSASensors* as a system type because its implementation will contain device instances to represent the particular sensors. For software-only services, we use a process type to indicate that the service will reside in its own address space, as shown in Figure 42. The *ASSASensors* system type has two feature groups as interfaces. *SensorSetOutput* represents the logical interface, which when configured will contain data ports of each different sensor reading. *SensorSetComm* represents the physical interface to the platform hardware; once configured it will indicate the necessary access requirements to networks (AADL bus access).

```
system ASSASensors
prototypes
  SensorSetOutput: feature group ASSAInterfaces::SensorTrackSets;
  SensorNetworkConnections: feature group ASSAInterfaces::SensorNetworkConnections;
features
  SensorSetReadings: feature group SensorSetOutput;
  SensorSetComm: feature group SensorNetworkConnections;
flows
  SensorReadings: flow source SensorSetReadings;
end ASSASensors;
```

Figure 41: Configurable ASSA Service

[9] For an overview of the NASA Mission Data System, see http://mds.jpl.nasa.gov/public.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

The interfaces of these services get configured for a particular aircraft platform in one of two ways (also see Section 6.4):

- The feature group type without features is specified as a *prototype*, identifying that any extension of the specified feature group type is acceptable as a configuration parameter. The prototypes are referred to in the feature group declaration (see Figure 41). We will supply an aircraft-specific feature group type, which is an extension of the feature group type referenced by the prototype, as a prototype actual when we declare the subcomponents of the service specification.
- The feature group type without features is referenced in the feature group declaration (see Figure 42). In this case, we will refine the feature group declaration of interest into one referencing the aircraft-specific feature group type in a process type extension.

```
process SAInformationPreparation
features
  WeatherInformation: in data port;
  AssessedInformation: in feature group inverse of ASSAInterfaces::AssessedTrackSets;
  ObstacleInformation: in data port;
  TerrainInformation: in data port;
  OwnAircraftPosition: in data port MissionSystemDataTypes::Position;
  MFDSAInformation: out feature group ASSAInterfaces::MFDSAInformation;
flows
  AssessedThreatInfo: flow path AssessedInformation -> MFDSAInformation;
  OwnAircraftInfo: flow path OwnAircraftPosition -> MFDSAInformation;
properties
  Period => 100 ms;
  Latency => 1 ms .. 1 ms applies to AssessedThreatInfo,OwnAircraftInfo;
end SAInformationPreparation;
```

Figure 42: Partial Service Specification

6.2 Reference Architecture Specification

We defined the reference architecture of ASSA in the package *ASSASystem::Common*. The specification of *ASSASystem* as a system type can be found in the package *ASSASystem* in the *SituationalAwarenessCommon* project. This specification defines the external interface of ASSA. It is an elaboration of the *ASSASystem* specification, as discussed in Section 3.3, that includes every external interface mentioned in the various MIS documents.

Figure 43 shows the reference architecture of ASSA as a system implementation. Each subcomponent represents a functional unit that is a configurable placeholder. The reference architecture includes connection declarations to represent the interaction topology within ASSA. Finally, the declaration includes a set of flow declarations that will be used in end-to-end flow analysis.

```
system implementation ASSASystem.Common
subcomponents
  Sensors: system ASSAServices::ASSASensors;
  Conversion: process ASSAServices::SADataConversion;
```


Distribution Statement A: Approved for Public Release; Distribution is Unlimited

```
  Fusion: process ASSAServices::DCFM;
  Assessment: process ASSAServices::SituationAssessment;
  ASSAFormatting: process ASSAServices::SAInformationPreparation;
  AirCrewDisplay: device ASSADisplayAnnunication::ASSAMFDDisplay;
  AircrewAnnunciation: device ASSADisplayAnnunication::ASSAMFDDisplay;
connections
  srctracks: feature group Sensors.SensorSetReadings -> Conversion.SensorSetReadings;
  stdtracks: feature group Conversion.StdSourceTracks -> Fusion.IncomingData;
  fusedtracks: feature group Fusion.OutgoingCorrelatedTracks -> Assessment.Assess-
mentInput;
  assessedtracks: feature group Assessment.AssessmentResults -> ASSAFormatting.As-
sessedInformation;
  formattedtracks: feature group ASSAFormatting.MFDSAInformation -> AirCrewDis-
play.MFDSAInformation;
  showtopilot: feature group AirCrewDisplay.AircrewSAInformation -> ASSAAirCrewPresen-
tation;
  ownaircraftposfusion: port OwnAircraftPosition -> Fusion.ownAircraftPosition;
  ownaircraftposdisplay: port OwnAircraftPosition -> ASSAFormatting.OwnAircraftPosi-
tion;
flows
  assessedASSAAircraftpos: flow path OwnAircraftPosition ->
        ownaircraftposfusion -> Fusion.AircraftPositionFusion ->
        fusedtracks -> Assessment.Assessment ->
        assessedtracks -> ASSAFormatting.AssessedThreatInfo ->
        formattedtracks -> AirCrewDisplay.ASSAInfoToPilot ->
        showtopilot -> ASSAAirCrewPresentation;
  directAircraftpos: flow path OwnAircraftPosition ->
        ownaircraftposdisplay -> ASSAFormatting.OwnAircraftInfo ->
        formattedtracks -> AirCrewDisplay.ASSAInfoToPilot ->
        showtopilot -> ASSAAirCrewPresentation;
  ASSASensorObservations: flow source Sensors.sensorReadings ->
        srctracks -> Conversion.TrackConversion ->
        stdtracks -> Fusion.TrackFusion ->
        fusedtracks -> Assessment.Assessment ->
        assessedtracks -> ASSAFormatting.AssessedThreatInfo ->
        formattedtracks -> AirCrewDisplay.ASSAInfoToPilot ->
        showtopilot ->ASSAAirCrewPresentation;
end ASSASystem.Common;
```

Figure 43: ASSA Reference Architecture

The context of the ASSA system is specified in a package called *AircraftSystem*. The system *AircraftSystem* has an implementation that consists of the ASSA system, an EGI to supply aircraft position, and the aircrew. We provided three variants of this system implementation. The first refers to the *ASSASystem* system type; it does not include the generic subsystems of the ASSA reference architecture specification. It is used in a system-level safety analysis. The second variant refines the original one to configure in the *ASSASystem* implementation with the subsystems. The

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

third variant refines the second variant to configure in a generic instance of the hardware platform for ASSA.

6.3 Reference Architecture Analysis

We defined end-to-end flows at the aircraft-system level to investigate potential time skew in displaying the own aircraft position by direct data flow from the EGI input to the display compared to the longer processing path and greater latency for display of threats or obstacles relative to own aircraft position. In addition, we can perform the response time analysis within the ASSA system by taking into account decisions about ARINC 653-compliant partition architecture, as we did in Sections 3.6.1 and 4.2.3. The specification of different services as AADL *process* provides a clear indication in the model that the service must be executed in a separate, runtime-enforced, protected address space.

We also use the reference architecture to perform a functional hazard assessment and fault impact analysis. Section 7 summarizes our approach, which we discuss in detail in a separate report titled *Architecture-Led Safety Analysis of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System* [Feiler 2015c].

6.4 Configuration for an Aircraft Platform

In this section, we describe how the reference architecture can be configured into an architecture for a specific aircraft platform. The example is illustrative only. We keep the aircraft-specific configuration specification in a separate set of packages in a folder called *ASSAConfigurations*. Configuration of the reference architecture involves several steps.

The first step is to elaborate the interfaces to a specific aircraft by defining extensions of various feature group types. The package *ASSAConfiguredInterfaces* contains these extensions. For the sensor interface, we introduce a data port for each sensor with a data type representing the sensor-specific data representation (see Figure 44). Similarly we configure the other interface specifications via the respective feature group type.

```
feature group SensorSourceTrackSets extends ASSAInterfaces::SensorTrackSets
features
  WWSensorTracks: out data port TrackTypes::WWTrackSet;
  ATWSensorTracks: out data port TrackTypes::ATWTrackSet;
end SensorSourceTrackSets;
```

Figure 44: Aircraft-Specific Interface Configuration

The second step is to elaborate the *ASSASensors* system into an aircraft-specific configuration by adding the appropriate types of sensors as instances (subcomponents), as shown in Figure 45. We define the system type of the configuration as an extension of the *ASSASensors* system in the reference architecture. As part of this specification, we supply the feature group types that are specific to the aircraft, also shown in Figure 45. In addition, we define the system implementation of *ASSASensorConfiguration*, which contains instances of the specific sensors for the target aircraft, connections to the *SensorSetReadings* feature group (the logical interface), and network access to the *SensorSetComm* feature group (physical interface). The package *ASSASensorConfigurations* contains these extensions.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

```
system ASSASensorConfiguration extends ASSAServices::ASSASensors
  (SensorSetOutput => feature group
     ASSAInterfaceConfigurations::CH47F::SensorSourceTrackSets,
   SensorNetworkConnections => feature group
     ASSAInterfaceConfigurations::CH47F::SensorNetworkConnections
  )
end ASSASensorConfiguration;

system implementation ASSASensorConfiguration.CH47F
subcomponents
  WW: device ASSASensors::WeaponsWatch;
  ATW: device ASSASensors::ATW;
connections
  WWconn : port WW.SourceTracks -> SensorSetReadings.WWSensorTracks;
  ATWconn : port ATW.SourceTracks -> SensorSetReadings.ATWSensorTracks;
  wwto1553 : bus access ww.to1553 -> SensorSetComm.To1553;
  atwto1553 : bus access atw.to1553 -> SensorSetComm.To1553;
end ASSASensorConfiguration.CH47F ;
```

Figure 45: Aircraft Sensor Configuration

The third step is to perform the same elaboration for the different ASSA services. The configuration for the SA data-conversion service can be found in the package *ASSAMISConfigurations* and is shown in Figure 46. The process type is extended to bind the appropriate feature group type configuration as prototype actual. The process implementation specifies a thread for each data-conversion function and connects it to the appropriate feature group port.

```
process ASSADataConversion extends ASSAServices::SADataConversion
( SensorTrackSets =>
   feature group ASSAConfiguredInterfaces::CH47FSensorSourceTrackSets,
  StdSourceTrackSets =>
   feature group ASSAConfiguredInterfaces::CH47FStdSourceTrackSets)
end ASSADataConversion;

process implementation ASSADataConversion.CH47F
subcomponents
  WWConversion: thread ASSADataConverters::WWTrackConverter;
  ATWConversion: thread ASSADataConverters::ATWTrackConverter;
 connections
  WWin: port SensorSetReadings.WWSensorTracks -> WWConversion.SourceTracks;
  WWout: port  WWConversion.StdSourceTracks -> StdSourceTracks.WWStdTracks;
  ATWin: port SensorSetReadings.ATWSensorTracks -> ATWConversion.SourceTracks;
  ATWout: port  WWConversion.StdSourceTracks -> StdSourceTracks.WWStdTracks;
end ASSADataConversion.CH47F ;
```

Figure 46: ASSA Data-Conversion Service Configuration

The final step is to configure the top-level system specification of the reference architecture into one for a specific aircraft. This configuration can be found in the package *CH47FConfiguration*. Figure 47 shows the configuration of the ASSA system by refining the specification of several subcomponents to the aircraft-specific specifications. In addition, we configured in an instance of a hardware platform and connected it to the ASSA system through a MIL-STD-1553 bus provided by the platform. We also configured the top-level specification for the aircraft system to utilize the aircraft-specific ASSA system configuration.

```
system implementation AircraftSystem.CH47F
    extends AircraftSystems::AircraftSystem.ASSASystem
subcomponents
  assa: refined to system ASSASystem.CH47F;
end AircraftSystem.CH47F;

system implementation ASSASystem.CH47F extends ASSASystem::ASSASystem.common
subcomponents
  Sensors: refined to system
    ASSASensorConfigurations::CH47F::ASSASensorConfiguration.CH47F;
  Conversion: refined to process
    ASSAServicesConfigurations::CH47F::ASSADataConversion.CH47F;
  Assessment: refined to process
    ASSAServicesConfigurations::CH47F::SituationAssessment.CH47F;
  Fusion: refined to process
    ASSAServicesConfigurations::CH47F::CorrelationFusion.CH47F;
  ASSAFormatting: refined to process
    ASSAServicesConfigurations::CH47F::SAInformationPreparation.CH47F;
  AirCrewDisplay: refined to device ASSAMFDDisplay;

  myplatform: system platform.ch47f;
connections
  sensorsto1553: feature group    Sensors.SensorSetComm ->
    myplatform.SensorNetworkAccess;
end ASSASystem.CH47F;
```

Figure 47: Aircraft Configuration

Once we completed the aircraft specific configuration, we could instantiate the aircraft system and revisit various analyses—in our case, the end-to-end latency analysis—to determine response time for a particular platform. We could also refine this architecture with different hardware platform configurations, partition configurations, and partition deployment configurations of the ASSA services to understand the impact of such deployment changes.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

7 ASSA System Safety Analysis Approach

From an aircraft-airworthiness perspective, the ASSA system is categorized as Design Assurance Level D, which means that the criticality level is minor. At the same time, enhancements to the ASSA system in the form of obstacle awareness have been justified by the fact that aircraft are lost more often due to collision with wires than due to enemy fire.

We examined the ASSA system from a safety perspective, including the following hazards. We will also do so on the ASSA reference architecture to demonstrate its feasibility.

- **Impact of lost ASSA service:** Analyzing risks to the pilot, aircraft, and mission by all the contributors to such losses gave us insight into the probability of their occurrence.
- **Incorrect SA data reporting (false positives and false negatives):** Incorrectly reporting the absence of tracked objects to the pilot would give the pilot a false sense of safety. For example, there is a risk that the absence of obstacles is really a failure of an ASSA function and that the pilot will not be aware of such a failure. This situation could result from mode confusion between the system and the pilot due to problems in the safety system, health monitor, or both.
- **Timeliness of SA presentation to pilot:** We consider whether latency contributors due to software are reflected in the error margins calculated and reported by DCFM.
- **Availability of ASSA services:** Unnecessary unavailability of ASSA can occur due to (1) overzealous mapping of exceptional conditions into fatal faults without attempt at recovery or repair and (2) the inability of the pilot to restart the ASSA service without completely rebooting the computing platform hosting ASSA and other services.

Given these hazards, we then identified potential hazard contributors. These are all failures of ASSA components and mismatched assumptions in the interactions between the components. Some of the hazard contributors are due to design decisions whose impact was not well understood. Those are avoidable hazards that can possibly be eliminated through changes in the design. Other hazard contributors are inherent, such as failure or malfunction of the physical SA sensor or the ASSA host computer. In these cases, we derived requirements for the SA health-monitoring system.

We annotated the model from the requirements specification task with fault information by using EMV2. The fault ontology, in terms of commonly occurring fault effect types, helped us consider various types of exceptional conditions that a subsystem failure can impose on interacting subsystems. In the process, we identified requirements for the SA health monitor in terms of what exceptional conditions it needs to detect or be informed of by subsystems, and what resulting systems states it needs to report to the pilot or automated system processing SA information. We summarize the annotated models and the results of this safety analysis in the *Architecture-Led Safety Analysis* report [Feiler 2015c].

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

8 Summary and Conclusion

The purpose of the ACVIP shadow project was to demonstrate the value of using ACVIP technology, in particular the value of using architecture models expressed in the SAE AADL standard, to discover potential system integration problems early in the development process. The SEI team captured information from existing requirements documents and other documentation as a requirements specification and architecture model expressed in AADL and a requirements specification notation. We then analyzed this system model for potential system integration issues.

This report summarized the ALRS process used to capture requirements and architecture specification of the JMR ASSA system in AADL and described the resultant model and analyses supported by the model. The ALRS process covers the 11 recommended practices of requirements specification outlined in the FAA *Requirement Engineering Management Handbook*. ALRS specifies requirements by focusing on the system with a well-defined boundary, its operational context, and its internal architecture; it utilizes system interface specifications, quality attribute utility trees, and a fault ontology to strive for improved requirements coverage.

We developed an AADL specification of the system in its operational context, as a functional architecture, and as a design architecture early in the development process. The resulting model reflected the architecture design information embedded in the requirements documents. In other words, it captured architecture decisions, whether they were made intentionally or unintentionally, and allowed us to assess the potential impact of these decisions.

We then performed a virtual integration of the system and an architecture analysis by investigating response times and architecture design decisions whose impact may not have been understood when specified. We also investigated the implications of a sequence diagram specification of the interaction between the MIS and the DCFM in the context of ARINC 653 partitioning by modeling the implementation of the protocols and determining their latency contributions analytically. We represented the resulting latency contributions as properties on the virtual bus abstraction of the protocol and used them to analyze the ASSA system. This protocol abstraction can also be used to analyze other application systems.

We annotated this AADL model with requirement specifications expressed in ReqSpec, a textual notation for the requirement specification subset of the RDAL metamodel. It identifies different parts of this system specification as a set of traceable requirements, whose satisfaction could be demonstrated by a set of verification actions. It also provides traceability to existing requirements documents and to stakeholders. By associating the requirements with the AADL model, we could assess whether the requirements address different elements of the ASSA system specification, such as interaction points with the operational context.

The AADL specification of a system, expressed as an AADL model, acts as a requirements specification in two ways. First, the customer or acquirer develops a specification, and the supplier response, also expressed as an AADL specification, is compared to the original to determine whether it meets the customer interest. Second, a specification represents a contract that a system implementation must meet. In this case, AADL models of the system architecture design, detailed design models in other notations, and code must be verified against this specification.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

We also showed how a configurable reference architecture for a situational awareness system can be specified in a way that makes it analyzable. This architecture model could be configured for specific aircraft platforms and re-analyzed.

By taking an architecture-led approach to specifying requirements for the ASSA system, the SEI team quickly identified a number of issues in the requirements documents for this system that, if not addressed, could result in system integration problems between MIS and DCFM. We documented these issues in a separate report [Feiler 2015a]. The issues include ambiguous, incomplete, and missing requirements; lack of clarity about the system boundary between MIS and DCFM; and mismatched assumptions in the interactions between MIS and DCFM. In addition, we found that architectural decisions, such as those reflected in the DCFM data model sequence diagrams, had unintended implications for the system's ability to meet response time requirements. Other architectural decisions created additional calibration requirements for DCFM where unexpected latency contributors and latency jitter introduced errors into track data.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Appendix Acronym List

AADL	Architecture Analysis & Design Language
ACVIP	Architecture-Centric Virtual Integration Practice
ALRS	Architecture-Led Requirements Specification
AMRDEC	Aviation and Missile Research, Development, and Engineering Center
ARINC	Avionics Application Standard Software Interface
ASSA	Aircraft Survivability Situation Awareness
ATAM	Architecture Tradeoff Analysis Method
BAA	Broad Agency Announcement
COP	Common Operational Picture
CoRE	Consortium Requirements Engineering
DCFM	Data Correlation and Fusion Manager
EGI	embedded GPS/inertial navigation system
EMV2	Error Model Version 2
FACE	Future Airborne Capability Environment
FENN	friendly, enemy, neutral and noncombatant
JCA	Joint Common Architecture
JMR	Joint Multi-Role
MFD	Multi-Function Display
MIS	Modular Integrated Survivability
NM	nautical miles
OSATE	Open Source AADL Tool Environment
RDAL	Requirements Definition & Analysis Language
SA	situational awareness
SEI	Software Engineering Institute
SSS	System/Subsystem Specification
STAMP	System-Theoretic Accident Model and Processes
QAW	Quality Attributes Workshop
UML	Unified Modeling Language
URI	Uniform Reference Identifier
WGS84	World Geodetic System 1984

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

References

URLs are valid as of the publication date of this document.

[AU 1996]

Air University, USAF Center for Strategy & Technology. 3.0 The Sensor and Fusion Process. In *New World Vistas: Air and Space Power for the 21st Century, Sensors Volume*. Federal Information Exchange. 1996. Pages 15–25. http://www.au.af.mil/au/awc/awcgate/vistas/sabmnse.htm

[BKCASE 2015]

Body of Knowledge and Curriculum to Advance Systems Engineering Editorial Board. *The Guide to the Systems Engineering Body of Knowledge (SEBoK)*. Version 1.4. Edited by R. D. Adcock. Trustees of the Stevens Institute of Technology. 2015. http://www.sebokwiki.org. Stakeholder Requirements: http://sebokwiki.org/wiki/Stakeholder_Needs_and_Requirements. System Requirements: http://sebokwiki.org/wiki/System_Requirements.

[Blouin 2011]

Blouin, D. et al. Defining an Annex Language to the Architecture Analysis and Design Language for Requirements Engineering Activities Support. In *Model-Driven Requirements Engineering Workshop (MoDRE), 2011*. Trento, Italy. August 2011. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6045362&filter=AND%28p_Publication_Number:6035928%29

[Boehm 2006]

Boehm, B. Some Future Trends and Implications for Systems and Software Engineering Processes. *Systems Engineering*. Volume 9. Number 1. January 2006. Pages 1–19. http://www.cs.cofc.edu/~bowring/classes/csis%20602/docs/FutureTrendsSEProcesses.pdf.

[de Niz 2012]

de Niz, Dio et al. *A Virtual Upgrade Validation Method for Software-Reliant Systems*. CMU/SEI-2012-TR-005. Software Engineering Institute, Carnegie Mellon University. 2012. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=10115

[FAA 2008a]

Federal Aviation Administration. *Requirements Engineering Management Handbook*. DOT/FAA/AR-08/32. FAA. 2008. http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/AR-08-32.pdf

[FAA 2008b]

Federal Aviation Administration. *Requirements Engineering Management Findings Report*. DOT/FAA/AR-08/34. FAA. 2008. http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/AR-08-34.pdf

[Faulk 1992]

Faulk, S. et al. The CoRE Method for Real-Time Requirements. *IEEE Software*. Volume 9. Number 5. September 1992. Pages 22–33.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

[Faulk 1993]
Faulk, S. et al. *Consortium Requirements Engineering Guidebook*. Technical Report SPC-92060-CMS. Software Productivity Consortium. 1993.

[Feiler 2009]
Feiler, Peter et al. *System Architecture Virtual Integration: An Industrial Case Study*. CMU/SEI-2009-TR-017. Software Engineering Institute, Carnegie Mellon University. 2009. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=9145

[Feiler 2010]
Feiler, Peter et al. *Case Study: Model-Based Analysis of the Mission Data System Reference Architecture*. CMU/SEI-2010-TR-003. Software Engineering Institute, Carnegie Mellon University. 2010. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=9407

[Feiler 2015a]
Feiler, Peter H. and Hudak, John. *Potential System Integration Issues in the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System*. CMU/SEI-2015-SR-030. Software Engineering Institute, Carnegie Mellon University. 2015. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=447176

[Feiler 2015c]
Feiler, Peter H. *Architecture-Led Safety Analysis of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System*. CMU/SEI-2015-SR-032. Software Engineering Institute, Carnegie Mellon University. 2015. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=447189

[Hayes 2003]
Hayes, J. H. Building a Requirement Fault Taxonomy: Experiences from a NASA Verification and Validation Research Project. Pages 49–59. In *14th International Symposium on Software Reliability Engineering (ISSRE)*. Denver, Colorado. November 2003. IEEE Computer Society Press, 2003.

[ISO 2011]
ISO/IEC/IEEE. Systems and software engineering - Requirements engineering. Geneva, Switzerland: International Organization for Standardization (ISO)/International Electrotechnical Commission/ Institute of Electrical and Electronics Engineers (IEEE), (IEC), ISO/IEC/IEEE 29148. 2011.

[ITU-T 2008]
International Telecommunication Union Telecommunication Standardization Sector. *Recommendation Z.151 (11/2008): User Requirements Notation (URN) – Language Definition*. ITU-T. November 2008.

[Leveson 2012]
Leveson, N. *Engineering a Safer World*. MIT Press. 2012.

[NIST 2002]
National Institute of Standards and Technology. *The Economic Impacts of Inadequate Infrastructure for Software Testing*. Planning Report 02-3. NIST. 2002.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

[Parnas 1995]
Parnas, D. and Madey, J. Functional Documents for Computer Systems. *Science of Computer Programming*. Volume 25. Number 1. October 1995. Pages 41–61.

[Rasmussen 2000]
Rasmussen, Jens and Svending, Inge. *Risk Management in a Dynamic Society*. Swedish Rescue Services Agency. 2000.

[Redman 2010]
Redman, David et al. Virtual Integration for Improved System Design. Pages 57–64. In *Proceedings of the First Analytic Virtual Integration of Cyber-Physical Systems Workshop in Conjunction with RTSS* 2010. San Diego, CA. November 2010.

[Schouwen 1990]
Van Schouwen, A. *The A-7 Requirements Model: Re-examination for Real-Time Systems and an Application to Monitoring Systems*. Technical Report 90-276. Queens University. 1990.

[SEI 2015]
Software Engineering Institute. *Latency Analysis*. Software Engineering Institute, Carnegie Mellon University. 2015. https://wiki.sei.cmu.edu/aadl/index.php/Latency_Analysis

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	December 2015	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Requirements and Architecture Specification of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System	FA8721-05-C-0003

6. AUTHOR(S)

Peter H. Feiler

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2015-SR-031

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFLCMC/PZE/Hanscom Enterprise Acquisition Division 20 Schilling Circle Building 1305 Hanscom AFB, MA 01731-2116	n/a

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

The Carnegie Mellon University Software Engineering Institute (SEI) was involved in an Architecture-Centric Virtual Integration Process (ACVIP) shadow project for the U.S. Army's Research, Development, and Engineering Command Joint Multi-Role program in the Joint Common Architecture (JCA) Demonstration. The JCA Demo used the Modular Integrated Survivability (MIS) system, which provides a situational awareness service that was integrated with two instances of a Data Correlation and Fusion Manager (DCFM) software component, which was contracted to two suppliers. The purpose of the ACVIP shadow project was to demonstrate the value of using ACVIP technology, in particular the architecture models expressed in the Society of Automotive Engineering Aerospace Standard 5506 standard for the Architecture Analysis & Design Language (AADL), for discovering potential system integration problems early in the development process. To do this, the SEI first captured information from existing requirements documents in AADL and the draft Requirement Definition & Analysis Language Annex. Then, by using an architecture-led approach to capturing requirements and architecture specification, the SEI team quickly identified a number of issues that, if not addressed, could result in system integration problems between MIS and DCFM. The SEI's findings allowed contractor teams to address these issues early in system development.

14. SUBJECT TERMS	15. NUMBER OF PAGES
AADL, Architecture-Centric Virtual Integration Practice, architecture models, software development, system integration, requirements specification	68

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18
298-102

Distribution Statement A: Approved for Public Release; Distribution is Unlimited