Software Engineering Institute
Carnegie Mellon University

Architecture-Led Safety Analysis of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System

Peter H. Feiler

December 2015

SPECIAL REPORT
CMU/SEI-2015-SR-032

Software Solutions Division

http://www.sei.cmu.edu

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.

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

DM-0002936

Table of Contents

List of Figures

Abstract

The Carnegie Mellon University Software Engineering Institute (SEI) team was involved in an Architecture-Centric Virtual Integration Process shadow project for the U.S. Army's Aviation and Missile Research, Development, and Engineering Center (AMRDEC) Science & Technology Joint Multi-Role (JMR) vertical lift program on the Joint Common Architecture (JCA) Demonstration. The JCA Demo used the Modular Integrated Survivability (MIS) system. The MIS project provided a Situational Awareness Data Manager service that was integrated with Data Correlation and Fusion Manager (DCFM). This report summarizes the approach taken in the architecture-led safety analysis of what will be referred to as the JMR Aircraft Survivability Situation Awareness (ASSA) system. The ASSA system was the focus of the Phase 2 MIS project, in which an AMRDEC team developed support services for ASSA and contractors provided a DCFM component. These components were implemented to conform to the Future Airborne Capability Environment (FACE) Standard specification for portability and integrated on two hardware platforms. By taking an architecture-led approach to safety analysis, the SEI team demonstrated the use of Architecture Analysis and Design Language and the Error Model V2 Annex standard in performing safety analysis of an embedded software system.

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 Aviation and Missile Research, Development, and Engineering Center (AMRDEC) Science and Technology Joint Multi-Role (JMR) program in the Joint Common Architecture (JCA) Demonstration. The JCA Demo used the Modular Integrated Survivability (MIS) system, which provides data management and health monitoring services for what we refer to as the Aircraft Survivability Situational Awareness (ASSA) system. This system was implemented through integration by an AMRDEC team of the MIS system with two instances of a Data Correlation and Fusion Manager (DCFM) software component. The DCFM was contracted out via a Broad Agency Announcement (BAA) to two suppliers.

The purpose of the JCA Demo ACVIP shadow project was to demonstrate the value of using ACVIP technology, in particular the use of architecture models expressed in the SAE Architecture Analysis and Design Language (AADL) standard in discovering potential system integration problems early in the development process. The SEI team applied the architecture-led requirement specification (ALRS) to capture requirements from existing requirement documents in models and identify potential integration issues early in development through virtual system integration and analysis. This aspect of the shadow project was discussed in the SEI special report *Requirement and Architecture Specification of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System* [Feiler 2015b]. The potential integration issues were reported in the SEI special report *Potential System Integration Issues in the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System* [Feiler 2015a].

Subsequent to the ALRS analysis, the SEI team then applied an architecture-led safety analysis (ALSA) to the ASSA by using the Error Model V2 (EMV2) standard to annotate the AADL model of ASSA with safety-related information. This document summarizes the approach taken in this safety analysis and describes the safety hazards and hazard contributors that have been identified, as well as the derived safety requirements for a Health Monitoring System (HMS) for ASSA.

The ASSA example illustrates the need for safety analysis of software-reliant systems. Software has been identified as a major source of hazards [Feiler 2009]. As recently as May 2015, software has been identified as the culprit—in this case it was found that the quad-redundant Generator Control Unit of the Boeing 787 could shut down all power during flight if operating for more than 248 hours [Goodin 2015]. These issues are not being identified by current best industry practices, such as SAE ARP 4761 [ARP 4761]. In many cases aircraft systems with embedded software are not considered flight critical and as a result are not assigned Design Assurance Level A. In the JCA Demo ACVIP shadow case study, the ASSA was assigned Design Assurance Level E. Current system safety analysis best practice is highly labor-intensive and as a result rarely repeated during the development process.

It has been demonstrated that automation of the system safety analysis practice SAE ARP 4761 is feasible through modeling and analysis with the SAE AADL and EMV2 standards [Delange 2014]. It is this technology that we use for the ALSA process described in this report.

2 An Architecture-Led Safety Analysis Process

The objective of the ALSA approach is to systematically identify hazards and hazard contributors in systems, in particular in embedded software systems, as they have become a major source of hazards. In this section we describe this process, which is centered on the use of AADL and Error Model V2 standards and integrated with an ALRS process.

2.1 Context of an Architecture-Led Safety Analysis

ALSA is performed in the context of a set of stakeholder and system requirement specifications as well as a socio-technical framework for hazard analysis. The *System Engineering Body of Knowledge* (SEBoK)[1] provides a classification of stakeholder requirements, shown in Figure 1, and of system requirements, shown in Figure 2. In both cases, the classification ranges from requirements for the system to requirements for development, such as business model and project constraints, and to requirements for operation, such as operational and logistical requirements, policy, and regulation. These categories are similar to elements of a general model of sociotechnical control, originally developed by Rasmussen [Rasmussen 2000] and adapted by Nancy Leveson of the Massachusetts Institute of Technology for the Systems-Theoretic Accident Model and Processes (STAMP) method of accident causality analysis [Leveson 2012].

[1] See http://www.sebokwiki.org/wiki/

Table 2. Example of Stakeholder Requirements Classification. (SEBoK Original)

Type of Stakeholder Requirement	Description
Service or Functional	Sets of actions to perform the mission or operation of the system-of-interest; enhanced by effectiveness or performance characteristics attached to the mission or operations.
Operational	This category may include: • Operational concepts that indicate the operational features to be provided without specifying design solutions. • Operational scenarios describing the sequence or series of activities supported by the system-of-interest. • Operational modes and transitions of modes between states/modes of the system-of-interest during its utilization to include dynamic interactions between the system-of-interest (viewed as a black box) and the system-of-interest's interface with external components in the context of its use.
Interface	Matter, energy, or information flows exchanged between the system-of-interest and its external components in the context of its use, including physical interfaces.
Environmental	External conditions that affect the system when in operation.
Utilization Characteristics	The '-ilities' used to indicate conditions of the utilization of the system-of-interest (e.g. usability, dependability, security, etc.).
Human Factors	Capabilities of users and operators, ergonomics, and associated constraints.
Logistical	Acquisition, transportation, and storage of elements needed by the system-of-interest to perform its services (e.g. constraints for logistical support).
Design and Realization Constraints	Reuse of existing system elements or forbidden materials, for example.
Process Constraints	These are stakeholder (usually acquirer or user) requirements imposed through the contract or statement of work. These requirements do not directly address the end-item system, but rather *how* the end-item system will be developed and provided. Process requirements include compliance with national, state, or local laws, such as environmental laws, administrative requirements, acquirer/supplier relationship requirements, and specific work directives. Process requirements may also be imposed on a program by corporate policy or practice. System or system element implementation process requirements, such as mandating a particular design method, are usually captured in project agreement documentation such as contracts, statements of work (SOW), and quality plans.
Project Constraints	Constraints to performing the project and/or the end-item system within cost and schedule.
Business Model Constraints	Constraints related to the expected business goal achieved by the system-of-interest, when this is relevant within the context of use, which may include: geographic position (local, national, international) of the future product, service, or organization resulting from the system-of-interest, distribution channels, alliance and partnership, finance and revenue model, etc.

Figure 1: SEBoK Stakeholder Requirement Classification

Table 2. Example of System Requirements Classification. (SEBoK Original)

Types of System Requirement	Description
Functional Requirements	Describe qualitatively the system functions or tasks to be performed in operation.
Performance Requirements	Define quantitatively the extent, or how well, and under what conditions a function or task is to be performed (e.g. rates, velocities). These are quantitative requirements of system performance and are verifiable individually. Note that there may be more than one performance requirement associated with a single function, functional requirement, or task.
Usability Requirements	Define the quality of system use (e.g. measurable effectiveness, efficiency, and satisfaction criteria).
Interface Requirements	Define how the system is required to interact or to exchange material, energy, or information with external systems (external interface), or how system elements within the system, including human elements, interact with each other (internal interface). Interface requirements include physical connections (physical interfaces) with external systems or internal system elements supporting interactions or exchanges.
Operational Requirements	Define the operational conditions or properties that are required for the system to operate or exist. This type of requirement includes: human factors, ergonomics, availability, maintainability, reliability, and security.
Modes and/or States Requirements	Define the various operational modes of the system in use and events conducting to transitions of modes.
Adaptability Requirements	Define potential extension, growth, or scalability during the life of the system.
Physical Constraints	Define constraints on weight, volume, and dimension applicable to the system elements that compose the system.
Design Constraints	Define the limits on the options that are available to a designer of a solution by imposing immovable boundaries and limits (e.g., the system shall incorporate a legacy or provided system element, or certain data shall be maintained in an online repository).
Environmental Conditions	Define the environmental conditions to be encountered by the system in its different operational modes. This should address the natural environment (e.g. wind, rain, temperature, fauna, salt, dust, radiation, etc.), induced and/or self-induced environmental effects (e.g. motion, shock, noise, electromagnetism, thermal, etc.), and threats to societal environment (e.g. legal, political, economic, social, business, etc.).
Logistical Requirements	Define the logistical conditions needed by the continuous utilization of the system. These requirements include sustainment (provision of facilities, level support, support personnel, spare parts, training, technical documentation, etc.), packaging, handling, shipping, transportation.
Policies and Regulations	Define relevant and applicable organizational policies or regulatory requirements that could affect the operation or performance of the system (e.g. labor policies, reports to regulatory agony, health or safety criteria, etc.).
Cost and Schedule Constraints	Define, for example, the cost of a single exemplar of the system, the expected delivery date of the first exemplar, etc.

Figure 2: SEBoK System Requirement Classification

2.2 An Integrated ALRS and ALSA Process

ALSA is performed in the context of the ALRS process, which has been discussed in another SEI special report [Feiler 2015b]. ALRS draws on the requirements engineering management 2009 handbook [FAA 2009]. The integrated ALRS/ALSA process borrows from several methods as appropriate. SEI Mission Thread Workshops help users identify and prioritize key mission drivers and use scenarios. SEI Quality Attribute Workshops focus on identifying and refining design and operational quality attributes into measurable properties about the system. The SEI Architecture Tradeoff Analysis Method (ATAM) evaluates architecture design against a prioritized set of mission drivers and use-case scenarios. The system safety analysis best practice (SAE ARP 4754 and 4761) describes Functional Hazard Assessment (FHA), Failure Mode and Effect Analysis (FMEA), Fault Tree Analysis, and Common Cause Analysis among others as methods to assess the safety of a system. STAMP is a causal hazard analysis method that focuses on system interactions as much as on failures of system components.

The integrated ALRS/ALSA process is shown in Figure 3. Safety analysis related steps are shown in green/italics.

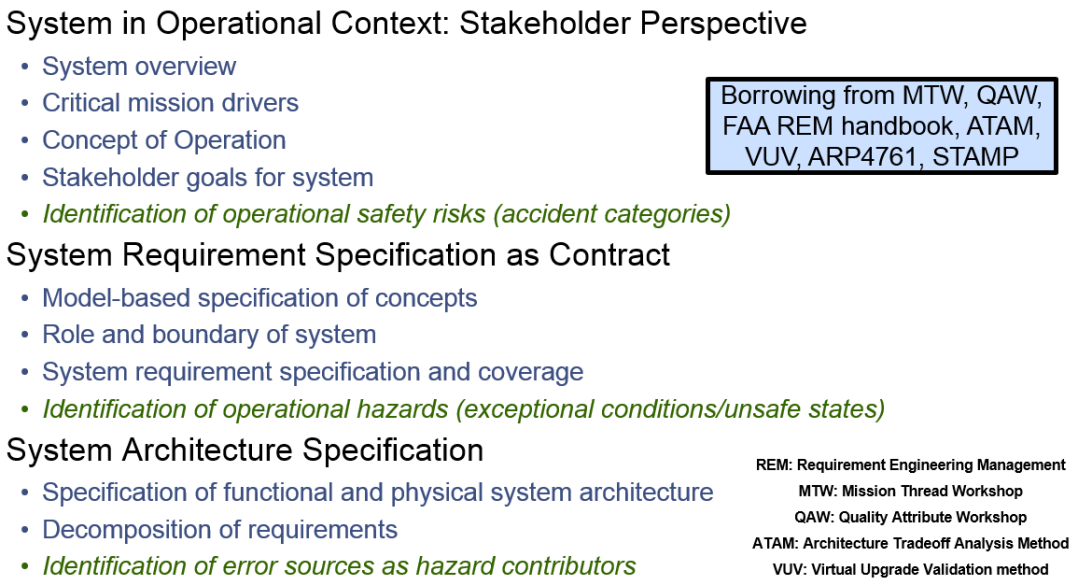

Figure 3: Integrated Architecture-Led Requirement Specification and Safety Analysis Process

The first aspect of this process focuses on capturing the stakeholder requirements. In the context of safety analysis, this is where operational safety risks are identified. They manifest themselves as a set of accident categories that are of concern to the stakeholders. In the case of ASSA, we identify entities in the operational environment of the aircraft whose unsafe interaction can lead to the loss of the aircraft.

The second aspect of this process focuses on specifying a set of system requirements that are verifiable (i.e., they represent a contract to be met). In that context, safety analysis identifies operational hazards that can lead to accidents of the previously identified accident categories. These hazards represent exceptional conditions that result in unsafe system behavior. These conditions may be failures of individual components, or they may be unsafe interactions between operational

components. In the case of ASSA, we recognize the role of ASSA as a system whose purpose is to inform the pilot of hazards in the operational environment and the hazards it presents when exhibiting faulty behavior.

The third aspect of this process focuses on identifying contributors to the identified hazards. This is done on an architecture specification of the system where each of the components is examined for failure conditions as well as component interactions that result in unsafe states. In the case of ASSA we identify failures in the functional and physical system architecture that can lead to hazardous presentation of information about hazards in the operational environment.

Once the contributors to hazards that lead to accidents are identified, the focus changes to strategies for mitigating the identified hazards and hazard contributors. This process is outlined in Figure 4.

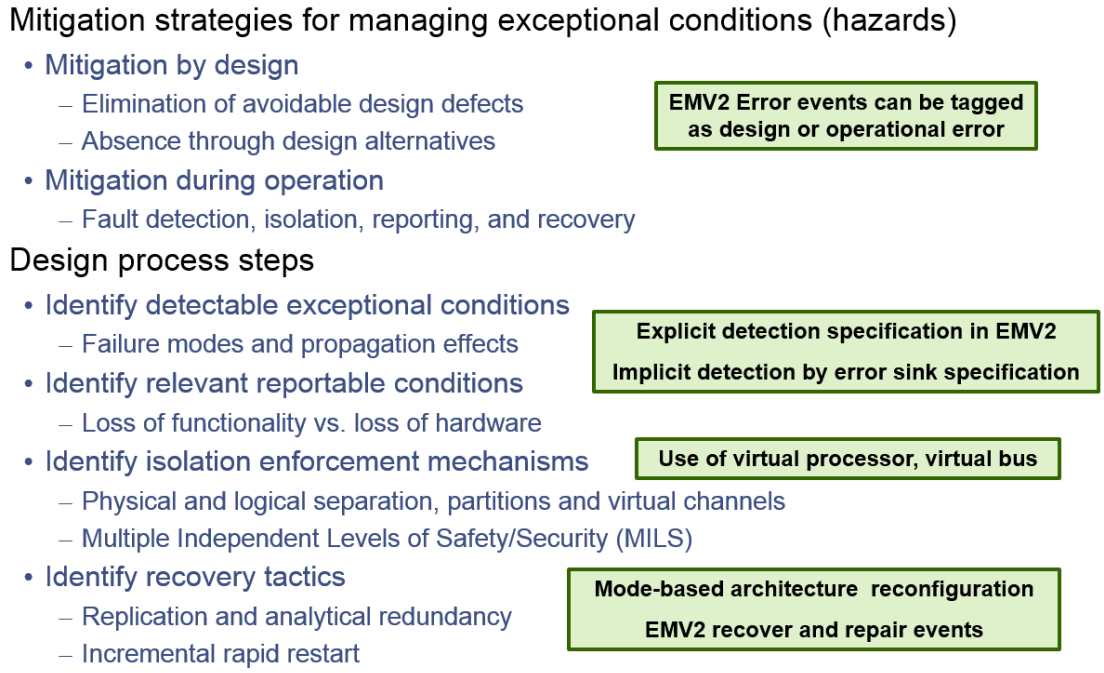

Figure 4: Hazard Mitigation Strategies in Designing a Safety System

Hazards may be the result of design defects that can be identified and removed during system design, or hazards may be unavoidable and, therefore, have to be mitigated during system operation. A safety system that mitigates hazards typically consists of one or more of the following elements —referred to as fault detection, isolation, reporting, and recovery:

- Fault detection is the capability of the safety system to monitor and detect faults during system operation. It may involve detecting the fault at its source, by monitoring downstream output and inferring upstream faults or by proactively probing the system through mechanisms such as heartbeat.
- Fault isolation is the capability of limiting the effect of a failure to a subset of the system. This is typically achieved by using fault containment mechanisms, such as address space protection in software, or sectioning of the power supply to different part of a system.

- Fault reporting is the capability to inform a system entity, which may be the system operator, of detected unsafe system behavior.
- Fault recovery is the capability to continue operation despite a failure through appropriate system redundancy, reconfiguration, and recovery mechanisms.

In the case of ASSA, we identify requirements for a health monitoring system that informs the pilot of the state of the ASSA.

2.3 Incremental ALRS/ALSA Process in ACVIP

The safety analysis and safety system design process continues incrementally along two dimensions—see Figure 5.

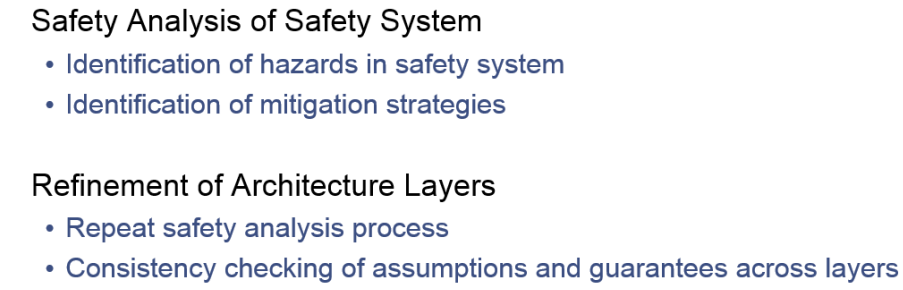

Figure 5: Two Dimensions of Incremental Safety Analysis and Safety System Design

The first dimension focuses on the safety system. It is analyzed for potential hazards, and mitigation strategies are developed. In the case of ASSA, we discovered that co-location of the health monitor software with the ASSA software on the same processor results in a common cause failure of the processor.

The second dimension of incremental safety analysis and safety system design is to iterate for each layer of the system architecture. From a system engineering perspective this is already part of the SAE ARP 4754A [ARP 4754A] recommended practice for civil aircraft and systems (Figure 6).

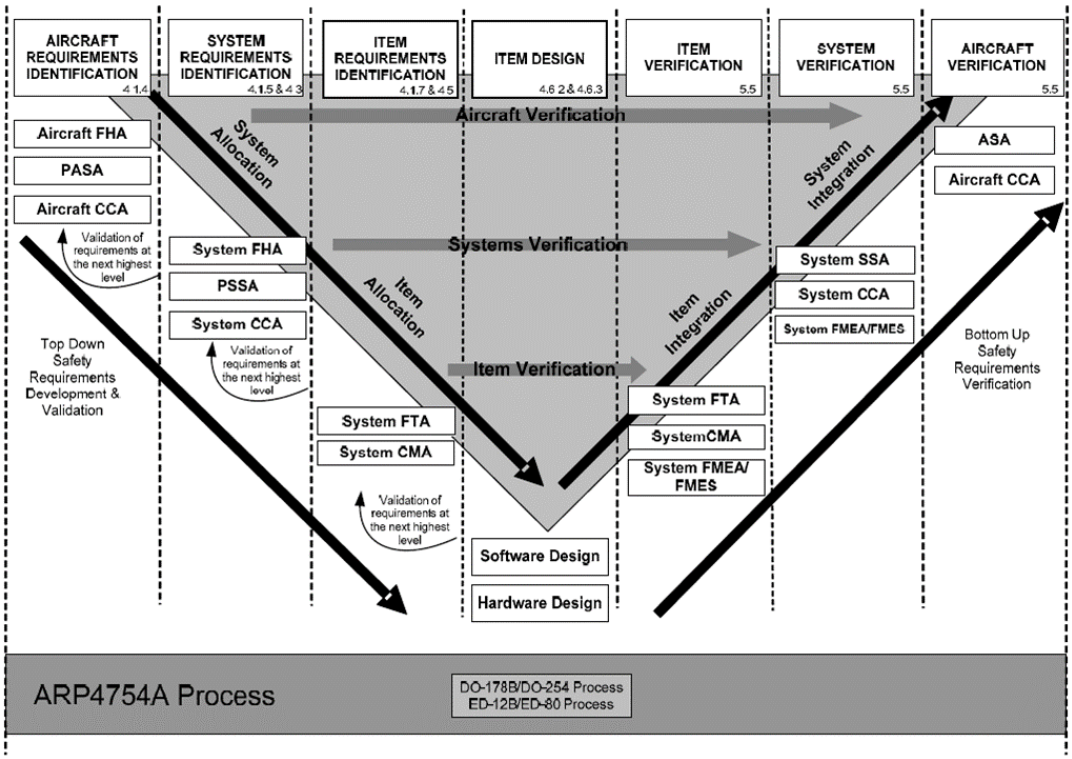

Figure 6: SAE ARP 4754A Guidelines for Development of Civil Aircraft and Systems [ARP 4754A]

Figure 7 highlights the safety practices within the ACVIP. The ACVIP consists of three major steps: define the operational context, develop the requirement specification, and develop the architecture specification. The safety process with ACVIP is a top-down approach conducted throughout the system hierarchy. It begins with the identification of operational safety risks (hazards) as part of defining the operational context for a system.

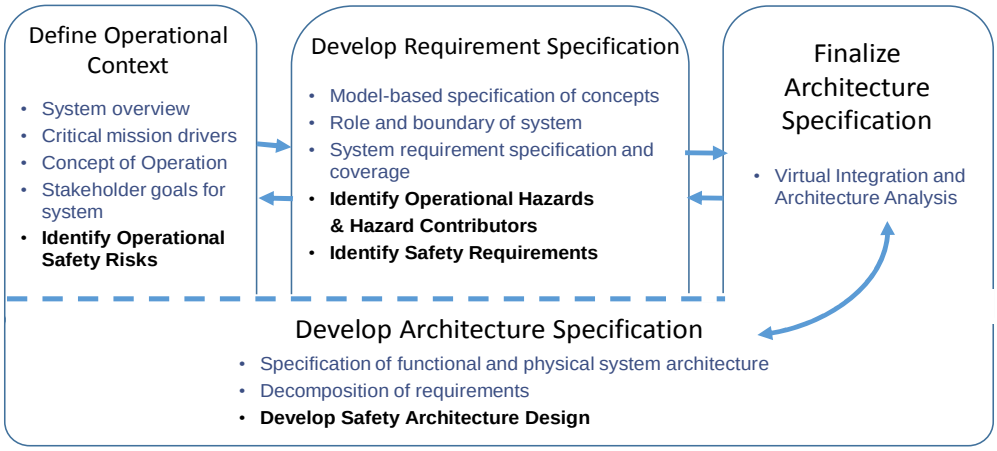

Figure 7: ACVIP ALRS/ALSA Process Steps

The process is iterative in that it is necessary to go back and make changes or additions to previous steps. The process is conducted through the various levels of the system and architectural hierarchy starting at the system level and continuing through to the component level of the architecture. This is shown in Figure 8. There is interplay and feedback among the various layers. First, the hazards, contributors, or requirements at a higher level are detailed in lower levels. Second, hazards, contributors, or requirements identified at one level may prompt the reorganization of a hazard, contributor, or requirement at a higher level. The execution of the process at a lower level may prompt the identification of a safety risk (hazard) at the top level. In Figure 8, *the Identify Operational Safety Risks* step is represented by opaque text at the more detailed hierarchical and component levels, indicating that the process is not explicitly conducted, since safety considerations relate to the complete system. The exceptional conditions (hazards) and contributors at lower levels are manifested as safety hazards at the system level.

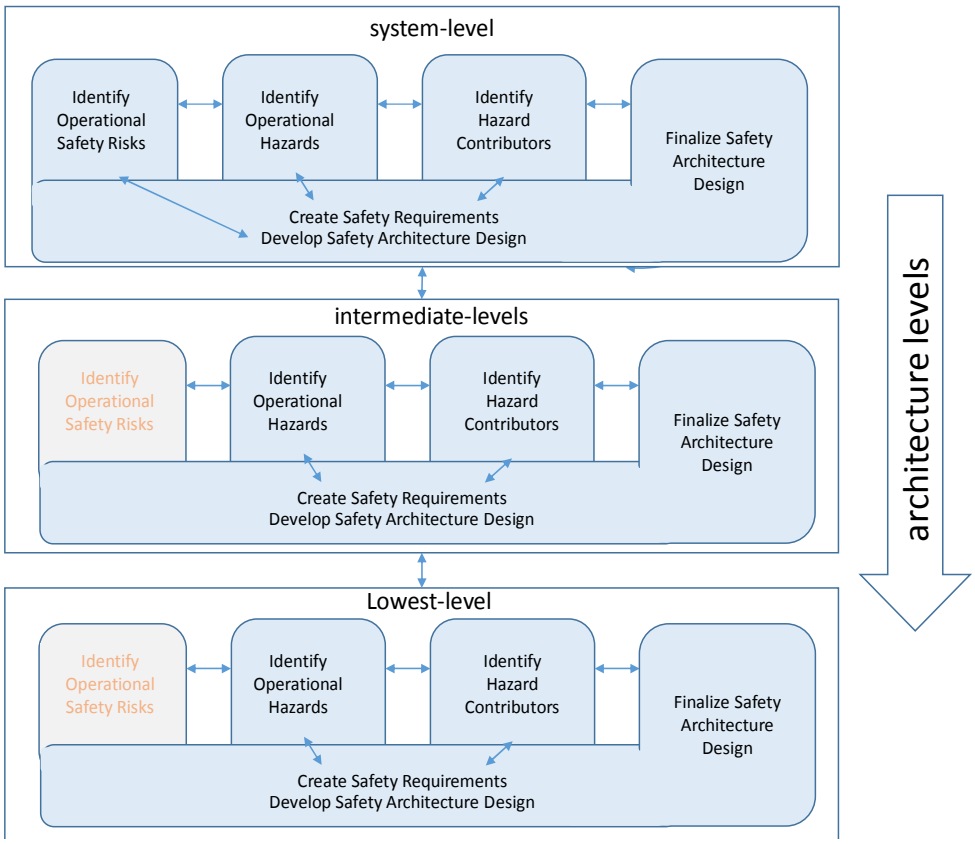

Figure 8: Iterations Through the System Hierarchy

The artifacts created during the process are shown in Figure 9. Hazards and their contributing factors (contributors) at multiple levels of the system hierarchy are identified. These are used as the basis for defining safety requirements for the system. These requirements are used to guide the overall system architecture design and may result in safety-specific architectural elements that are incorporated into the system architecture.

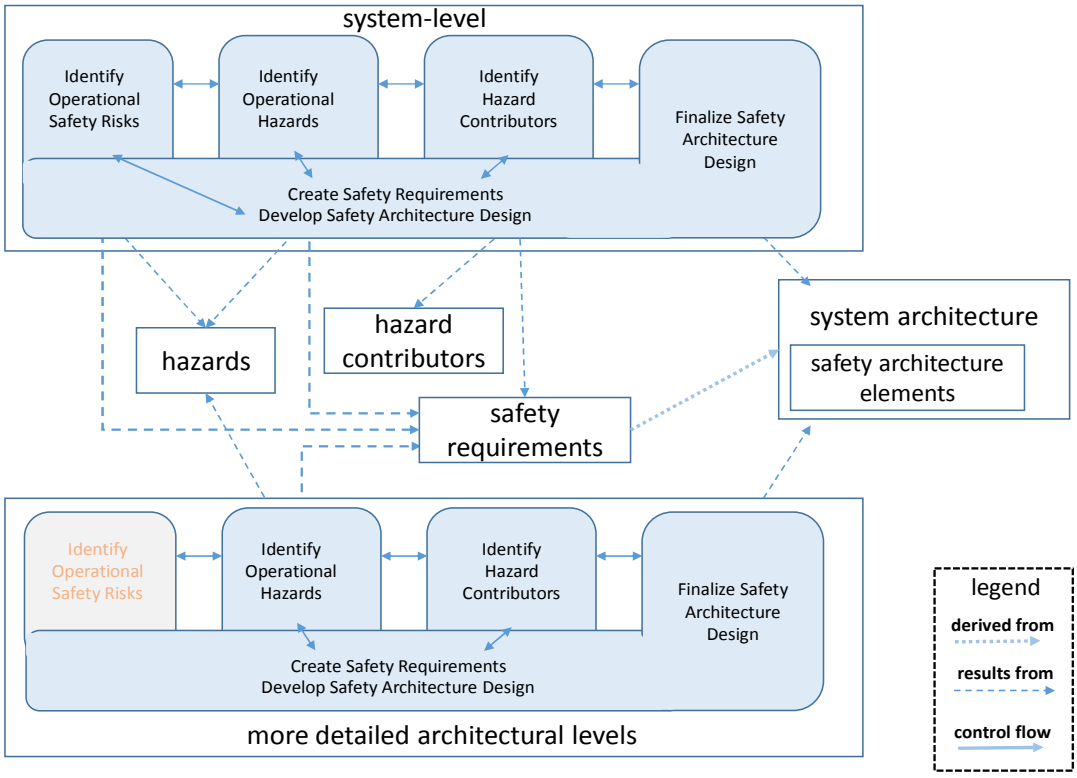

Figure 9: Process Artifacts

2.4 Requirement and Hazard Coverage

We improve the quality of requirement specification by providing a measurable way of assessing requirement coverage. This consists of three parts:

1. Identification of 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 is expected to be addressed by requirements.
2. Identification and quantification of design and operational quality attributes that are key to achieving the mission. Each of these key quality attributes must be addressed by a requirement specification.
3. Identification of 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 to other systems and that other systems potentially propagate to the system of interest.

2.4.1 Coverage of System Interaction and Behavior

We use a framework for specifying a system that has its origin with the French System Engineering Society (*Association Française d'Ingénierie Système*).[2] This framework, called Constraints,

[2] See http://en.wikipedia.org/wiki/Process(engineering)

Products, Resources, Elements, and Transformation (CPRET), is illustrated in Figure 10. A system is defined to transform a set of inputs into a set of outputs while potentially maintaining state, utilizing resources, and being under control of a supervisory entity.

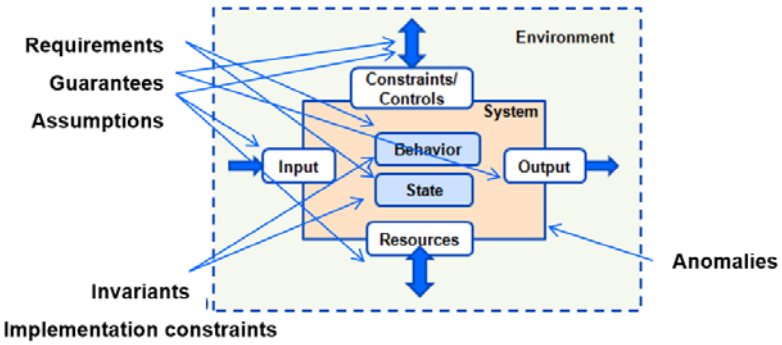

Figure 10: Constraints, Products, Resources, Elements, and Transformation (CPRET)

Each interaction point is expected to be addressed by requirements. The specification of each interaction point is expected to 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 must be considered. For input, supervisory control, and resource usage interaction points this represents assumptions being made about the operational environment. For output and supervisory control feedback this represents guarantees made by the system to others.

2.4.2 Coverage of Relevant Design and Operational Quality Attributes

Next we utilize the concepts of quality attributes and utility trees from the SEI ATAM. These quality attributes represent two categories of requirements:

1. developmental requirements, such as modifiability, portability, or assurability
2. operational requirements, with subcategories of mission-, safety-, and security-critical requirements. Mission-critical requirements include function, behavior, and performance. Safety-critical requirements deal with mitigating hazards. And security-critical requirements deal with assuring protection of information and trust.

Figure 11 illustrates a partial set of quality attributes—three operational, and one developmental. Figure 11 also shows a refinement of the quality attributes into utility functions and their quantification into requirements, whose satisfaction can be measured. The annotations of L(ow)/M(edium)/H(igh) pairs indicate levels of criticality and difficulty to help focus architectural design and evaluation/verification.

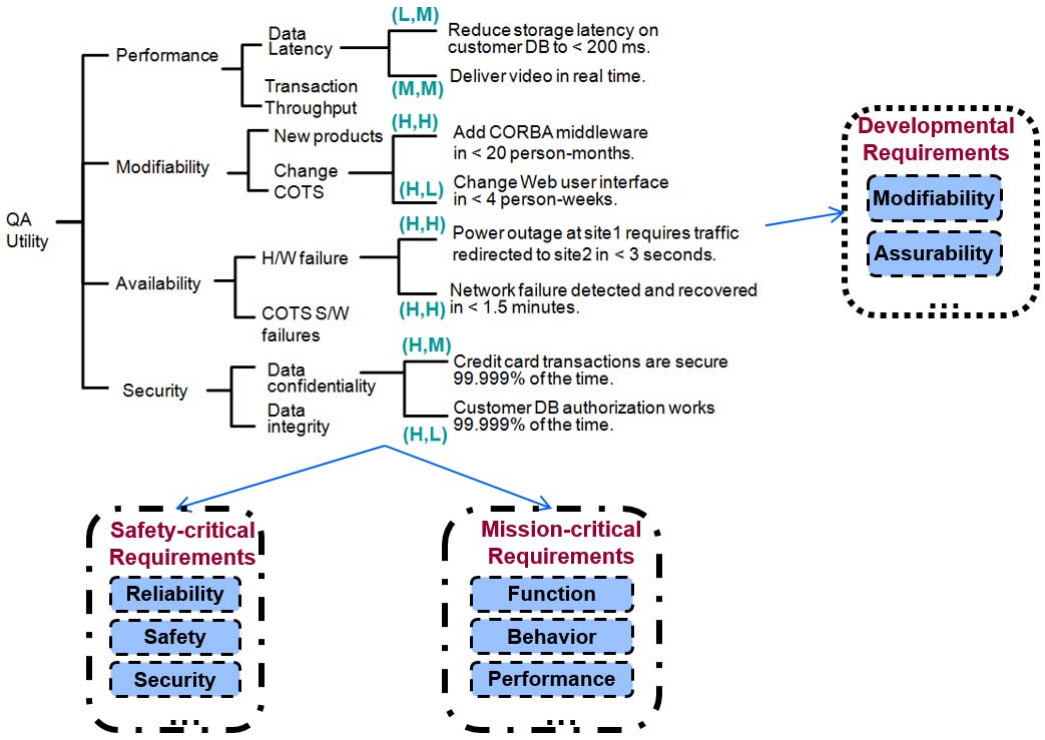

Figure 11: Operational Quality Attributes and Utility Trees

This utility tree becomes a checklist for assuring that relevant quality attributes of the system are being addressed by requirement specifications.

2.4.3 Coverage of Faults and Their Impact

Finally, we use a fault ontology that has been defined as part of the EMV2 language standard that is part of the SAE AADL standard suite. Figure 12 illustrates the fault ontology on the left. This ontology focuses on propagating effects of system failure modes to other systems. The most common effect is omission (i.e., the failure to provide a service or output). An example is failure to provide power. Commission is when service or output is provided at a time when it is not expected. Other examples of the ontology are value errors, timing errors, replication errors, and concurrency errors. (For more on the fault ontology and its interaction with hazard and operability (HAZOP) guide words, see Section 3.1.)

The right side of Figure 12 illustrates the systematic application of these error types to a system specification. It shows the interaction between a control system and a system under control. We annotate this specification with error types to indicate whether certain faults are expected to occur.

STAMP 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 ATAM, leading to classifications that tend to align with the EMV2 fault ontology.

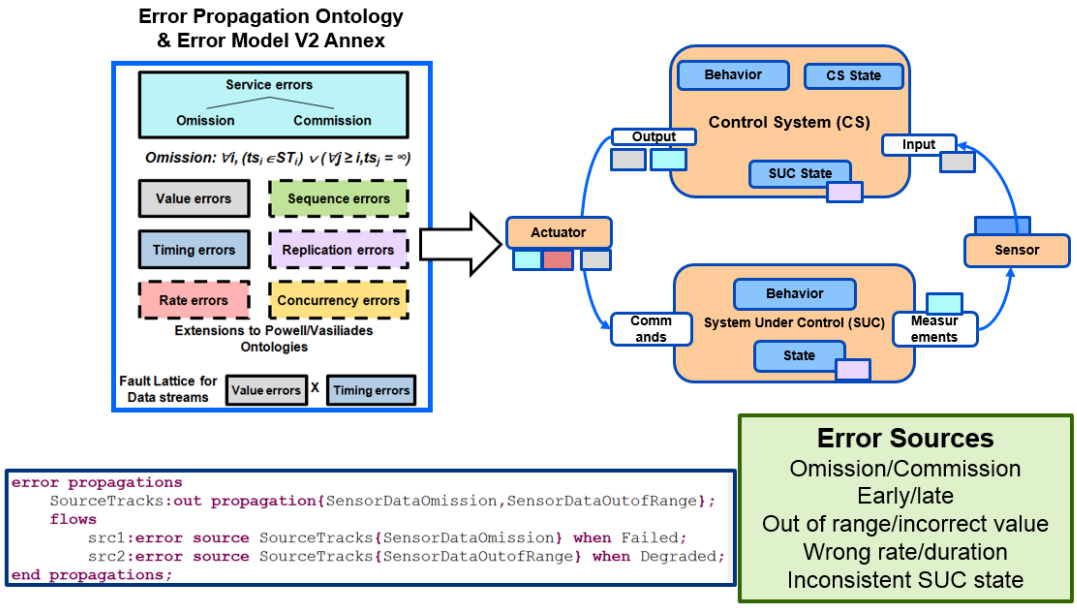

Figure 12: Fault Ontology and Its Application to a System Specification

3 Modeling with AADL and the Error Model V2 Standard

To create AADL models, we use 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. The SEI special report *Requirement and Architecture Specification of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System* provides additional guidance on how to capture a system in its operational context, stakeholder and system requirements, and the system architecture in an ACVIP manner [Feiler 2015b].

In this document, we summarize how to use the EMV2 standard to support ALSA. A full guide on the use of EMV2, titled *Architecture Fault Modeling and Analysis with the Error Model Annex V2*, will be available as an SEI special report.[3]

We use EMV2 to

- systematically identify exceptional conditions that, when propagated to other systems and system components, represent hazards. We use
  - the EMV2 fault ontology expressed as EMV2 error types that act as a check list or HAZOP style guide words
  - EMV2 error propagation declarations to specify outgoing propagations that are error sources
- systematically address how systems respond to incoming propagations. We use
  - incoming error propagation declarations to specify that a system expects error propagations from other system components
  - error sink and path declarations to specify that incoming propagations are masked (e.g., extrapolate a missing value from previous values), passed on to other components (e.g., produce no output if input is missing), or transformed to a different error type (e.g., send no output if the input is out of range or otherwise corrupted)

We proceed by first elaborating on the EMV2 fault ontology and then describing the EMV2 constructs to specify error propagation behavior across the system.

Finally, as we focus on derived requirements for the ASSA health monitor, we use EMV2 declarations to specify assumptions about a safety system (e.g., who is responsible for detection of fault occurrences) and the effect of recovery actions by the safety system on the error states representing working states and failure modes.

[3] Feiler, Peter H. et al. *Architecture Fault Modeling and Analysis with the Error Model Annex V2*. Special Report. Software Engineering Institute, Carnegie Mellon University. Forthcoming.

3.1 EMV2 Fault Ontology and Guide Words

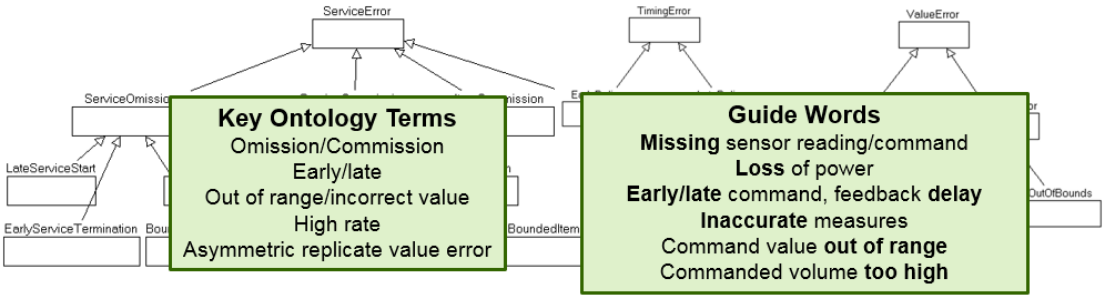

Figure 13: EMV2 Fault Ontology and Guide Words

The EMV2 fault ontology focuses on characterizing error propagation between system components (i.e., the effect a fault occurrence in one component can have on other components). Common error propagation types are shown in Figure 13, while the full ontology is described in subsections 3.1.1 through 3.1.6. The intent of the ontology is to provide a checklist of error propagation types in abstract terms that can be adapted to specific domains and applications. Its role is similar to guide words in a HAZOP process (example shown in Figure 14).

Parameter / Guide Word	More	Less	None	Reverse	As well as	Part of	Other than
Flow	high flow	low flow	no flow	reverse flow	deviating concentration	contamination	deviating material
Pressure	high pressure	low pressure	vacuum		delta-p		explosion
Temperature	high temperature	low temperature					
Level	high level	low level	no level				
Time	too long / too late	too short / too soon	sequen step skipped				
Agitation	fast mixing	slow mixing	no mixi				
Reaction	fast reaction / runaway	slow reaction	no reactior				
Start-up / Shut-down	too fast	too slow					

Guide Word	Meaning
NO OR NOT	Complete negation of the design intent
MORE	Quantitative increase
LESS	Quantitative decrease
AS WELL AS	Qualitative modification/increase
PART OF	Qualitative modification/decrease
REVERSE	Logical opposite of the design intent
OTHER THAN	Complete substitution
EARLY	Relative to the clock time
LATE	Relative to the clock time
BEFORE	Relating to order or sequence
AFTER	Relating to order or sequence

Figure 14: HAZOP Guide Word Tables

The EMV2 fault ontology is expressed as a set of error types that are then used to characterize error propagations. The error types are defined by viewing components as providers of services that consist of a sequence of service items. Error types fall into the categories of service-related errors, value-related errors, time related errors, and errors related to redundancy and concurrency. Furthermore, within each category the error types may characterize the service as a whole, the sequence of service items, or an individual service item. The EMV2 Annex standard includes a formal specification of each of the error types. The EMV2 Annex standard also includes a user-extensible set of aliases for some of the error types that reflect application-specific guide words.

3.1.1 Service-Related Errors

Service-related errors (*ServiceError*) represent errors with respect to the number of delivered service items. We distinguish between omission errors to represent service items not delivered and commission errors to represent delivery of service items that were not expected to be delivered.

The error types for individual service items as subtypes of *ServiceError* are

- *ItemOmission* (i.e., the omission of a single service item such as a lost message)
- *ItemCommission* (i.e., provision of an item when not expected such as a spurious message)

The error types for a sequence of service items (*SequenceOmission*) are

- *SequenceOmission* (i.e., a number of missing service items, such as missed sensor readings)
    - *BoundedOmissionInterval* (i.e., a minimum number of service items between item omissions such as missed sensor readings)
    - *TransientServiceOmission* (i.e., a limited sequence of item omissions such as a temporary power outage)
    - *EarlyServiceTermination* (i.e., omission of all service items partway into the service provision such as a power failure)
    - *LateServiceStart* (i.e., initial service items not provided such as difficulty in starting a generator to provide power)
- *SequenceCommission* (i.e., a limited sequence of item commissions with the following subtypes):
    - *TransientServiceCommission* (i.e., a limited sequence of item extra service items, such as extra alarm messages)
    - *LateServiceTermination* (i.e., additional service items after the expected termination of service, such as warning messages about an overheated engine after the engine stops)
    - *EarlyServiceStart* (i.e., extra service items are provided before the expected service start, such as sensor readings before engine start)

The error types for the service as a whole are

- *ServiceOmission* (i.e., failure to provide a service when expected such as no power due to blown transformer)
- *ServiceCommission* (i.e., provision of service when not expected such as inadvertent charge on an inactive power line)

These errors have been placed into a type hierarchy shown in Figure 15.

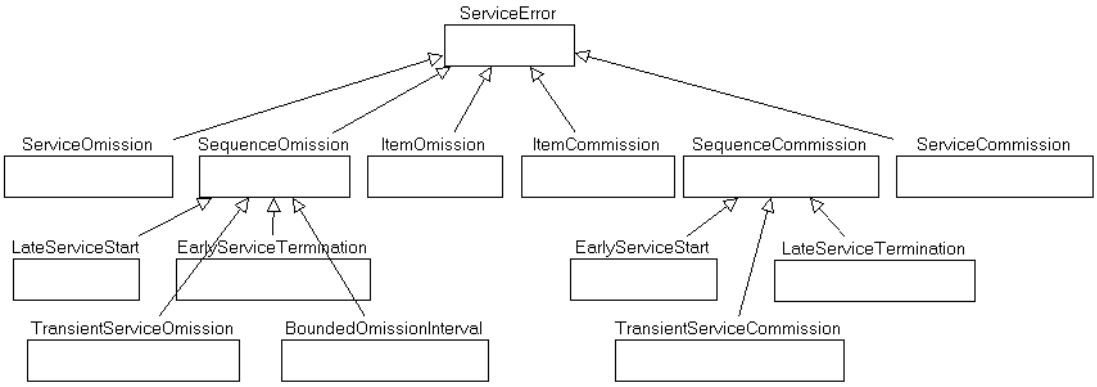

Figure 15: Service-Related Error Type Hierarchy

3.1.2 Value-Related Errors

Value-related errors deal with the value domain of a service. We distinguish between value errors of individual service items (*ItemValueError*), value errors that relate to the sequence of service items (*SequenceValueError*), and value errors related to the service as a whole (*ServiceValueError*). They form the type set *ValueRelatedError*.

Each of the three types is the root of a separate type hierarchy. This allows us to use them in combination in a type product (e.g., to specify that we have a *BoundedValueChange* error that may be *OutOfRange*).

ItemValueError consists of

- *DetectableValueError* (i.e., a value error that is detectable from the value itself, perhaps because it is out of range or has parity error)
- *UndetectableValueError* (i.e., a value error that cannot be recognized based on available information)

DetectableValueError has the following subtypes

- *OutOfRange* error (i.e., a value that is outside a specified range), with two subtypes—*BelowRange* and *AboveRange*
- *OutOfBounds* error (i.e., a value error that may be within range, but whose value affects a state in such a way that it will be outside specified bounds). For example, in a control system a command to move a certain number of steps may be within range of the maximum number of steps that can be executed in a frame, but may result in a position that is outside the range of acceptable positions.

SequenceValueError consists of

- *BoundedValueChange* (i.e., the difference between two consecutive values is greater than a specified limit). For example, in a control system set-point values may be expected to only change by up to a specified value.
- *StuckValue* (i.e., a value that remains the same for a number of consecutive service items)
- *OutOfOrder* (i.e., values in the sequence that are not in the correct order)

ServiceValueError consists of

- *OutOfCalibration* (i.e., a value error where all values are off by some value). For example, in a control system due to an incorrect calibration value all controller output values are not correct.

Figure 16 shows the type hierarchies for value-related errors. Note that the top-level error types are grouped into the type set *ValueRelatedError* (not shown graphically). Note also that both sequence and service value errors imply item value errors.

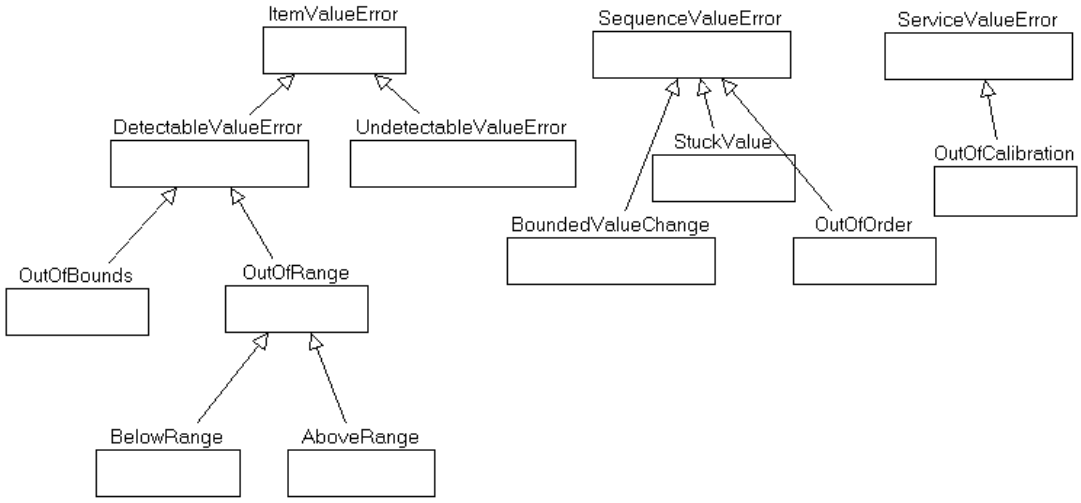

Figure 16: Value-Related Error Type Hierarchy

The EMV2 Annex standard includes a predeclared set of aliases for value errors, as shown in Figure 17.

```
-- Common aliases for value related errors
ValueError renames type ItemValueError; -- legacy
SequenceError renames type SequenceValueError; -- legacy

IncorrectValue renames type ItemValueError;
ValueCorruption renames type ItemValueError;
BadValue renames type ItemValueError;

SubtleValueError renames type UndetectableValueError;
BenignValueError renames type DetectableValueError;
BenignValueCorruption renames type DetectableValueError;
SubtleValueCorruption renames type UndetectableValueError;
```

Figure 17: Aliases for Value-Related Error Types

3.1.3 Timing-Related Errors

Timing-related errors deal with the time domain of a service. We distinguish between timing errors of individual service items (*ItemTimingError*), timing errors that relate to the sequence of service items (*SequenceTimingError* or alias *RateError*), and timing errors regarding the service as a whole (*ServiceTimingError*). They form the type set *TimingRelatedError*.

Each is the root of a separate type hierarchy allowing us to characterize them independently (e.g., to specify that we have a time shifted service executing at the wrong rate). Item timing errors and sequence timing errors refer to a timeline with respect to service start time, while service timing errors use clock time as reference time. Therefore, service timing errors are independent of the other two, while sequence timing errors imply item timing errors.

ItemTimingError consists of

- *EarlyDelivery* (i.e., delivery of a service item before an expected time range, such as a sensor reading arriving before the previous reading has been sampled for processing)
- *LateDelivery* (i.e., delivery of a service item after an expected time range, such as a sensor reading arriving after the beginning of the next frame)

SequenceTimingError with the alias *RateError* consists of

- *HighRate* error (i.e., the inter-arrival time of all service items is less than the expected inter-arrival time). For example, a sender sends periodic messages with a period of 25ms, while the receiver processes the messages as they arrive and takes an average of 26ms to complete processing.
- *LowRate* error (i.e., the inter-arrival time of all service items is greater than the expected inter-arrival time)
- *RateJitter* error (i.e., service items are delivered at a rate that varies from the expected rate by more than an acceptable tolerance)

ServiceTimingError with the alias *ServiceTimeShift* represents errors where a service delivers all service items time shifted by a time constant. It consists of two subtypes—*DelayedService* and *EarlyService*.

The type hierarchies for timing-related errors are shown in Figure 18. Note that the top-level error types are grouped into the type set *TimingRelatedError* (not shown in Figure 18).

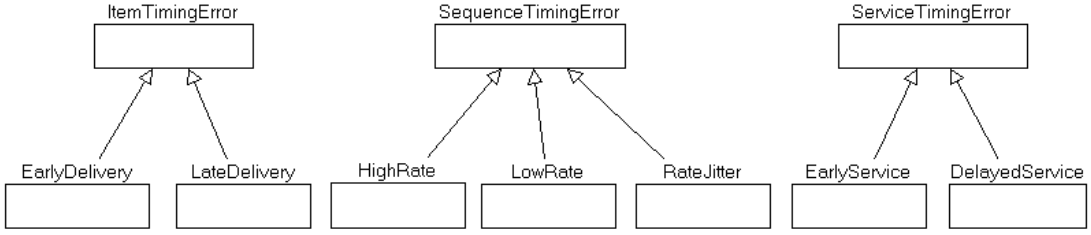

Figure 18: Timing-Related Error Type Hierarchy

The EMV2 Annex standard includes a predeclared set of aliases for timing errors, as shown in Figure 19.

```
TimingError renames type ItemTimingError; -- legacy
RateError renames type SequenceTimingError; -- legacy
EarlyData renames type HighRate;
LateData renames type LowRate;
ServiceTimeShift renames type ServiceTimingError;
```

Figure 19: Aliases for Timing-Related Error Types

3.1.4 Replication-Related Errors

Replication-related errors (*ReplicationError*) deal with replicates of a service item. Replicate service items may be delivered to one recipient (e.g., a fault tolerance voter mechanism) or to multiple recipients (e.g., separate processing channels). Replicate service items may be the result of inconsistent fan-out from a single source, or they may be the result of an independent error occurring to individual replicates (e.g., readings of the same physical entity by multiple sensors or an error occurrence in one of the replicated processing channels).

ReplicationError consists of

- *AsymmetricReplicatesError* (i.e., at least one of the replicates is different from the others)
- *SymmetricReplicatesError*, where all replicates have the same error (e.g., the error was introduced before the service item was replicated)

We distinguish between the following asymmetric replicates errors:

- *AsymmetricValue* error with the alias *InconsistentValue* (i.e., the value of at least one replicated service items differs from the other replicates). In the case of the subtype *AsymmetricExactValue* error, the values are expected to be exactly the same, while for the subtype *AsymmetricApproximateValue* they cannot differ by more than a threshold.
- *AsymmetricOmission* error with the alias *InconsistentOmission* with the subtype *AsymmetricItemOmission* (i.e., at least one of the replicates is missing) (encounters an *ItemOmission*) and *AsymmetricServiceOmission* (i.e., at least one of the replicates is missing) (encounters a *ServiceOmission*)
- *AsymmetricTiming* error with the alias *InconsistentTiming* (i.e., at least one of the replicated service items is delivered outside the expected time interval)

We have the respective set of error subtypes for *SymmetricReplicatesError*.

Figure 20 illustrates the type hierarchy for replication errors.

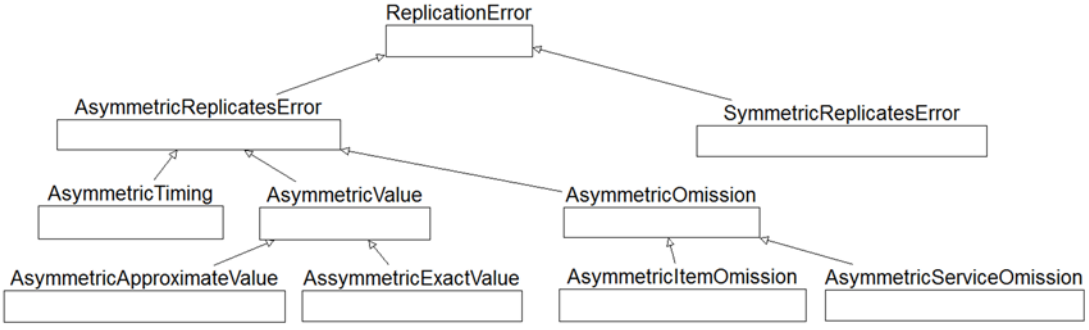

Figure 20: Replication Error Type Hierarchy

The EMV2 Annex standard includes a predeclared set of aliases for replication errors as shown in Figure 21.

```
InconsistentValue renames type AsymmetricValue;
InconsistentTiming renames type AsymmetricTiming;
InconsistentOmission renames type AsymmetricOmission;
InconsistentItemOmission renames type AsymmetricItemOmission;
InconsistentServiceOmission renames type AsymmetricServiceOmission;
AsymmetricTransmissive renames type AsymmetricValue;
```

Figure 21: Aliases for Replication Error Types

3.1.5 Concurrency-Related Errors

Concurrency-related errors (*ConcurrencyError*) address issues that occur when concurrently executing tasks access shared resources. We distinguish between race conditions (*RaceCondition*) in the form of *ReadWriteRace* and *WriteWriteRace*, and mutual exclusion errors (*MutExError*) in the form of *Deadlock* and *Starvation*. Figure 22 shows the type hierarchy.

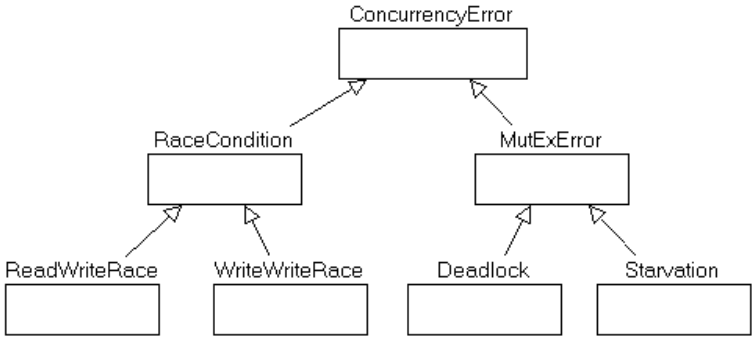

Figure 22: Concurrency Error Type Hierarchy

3.1.6 Authorization- and Authentication-Related Errors

Authorization-related errors (*AuthorizationError*) are related to access control. Authorization errors consist of privilege enforcement errors and privilege administration errors. Examples of authorization errors are ambient authority errors, privilege escalation errors, confused deputy errors, privilege separation errors, privilege bracketing errors, compartmentalization errors, least privilege errors, privilege granting errors, and privilege revocation errors.

Authentication-related errors (*AuthenticationError*) are related to authentication of services (roles, agents), of information, and of resources.

3.2 Fault Propagation Across the System

With EMV2 we can annotate individual system components with outgoing and incoming error propagations and whether they act as a source, sink, or pass-through of fault occurrences. Combined with the propagation paths already specified in the AADL model in terms of port connections, access connections, and deployment bindings (shown in Figure 23), we have all the elements to support the Fault Propagation and Transformation Calculus (FPTC) [Paige 2009].

This allows a tool to perform inductive and deductive fault impact analysis (i.e., identify effects of fault occurrences and identify contributors to hazards or accidents). In the context of the case study, we are able to identify hazard contributors of ASSA.

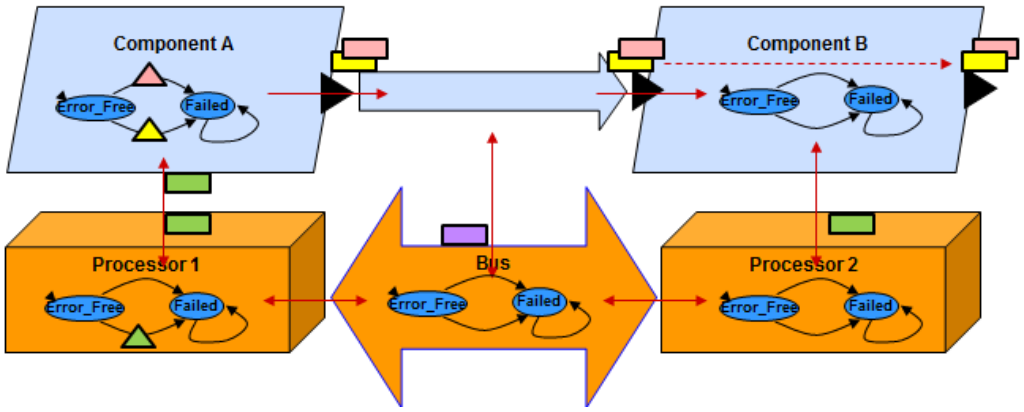

Figure 23: Error Propagation Across the System

The following EMV2 concepts are used to annotate system components:

- *error propagation* and *containment* associated with interaction points (ports, data and bus access, remote service calls, deployment binding points) to other components to specify the different types of effect, such as bad value or no service, a component failure or incoming propagation can have on other components, or that a component is expected not to propagate certain types of effects. Note that outgoing and incoming propagation and containment specifications act as contracts between interacting components (i.e., as guarantees and assumptions that must be verified).
- *error types* for characterizing the different types of errors being propagated (e.g., a value error or timing error) or different types of error events (e.g., a component being overheated, cracked, or stuck)
- *error sources* for identifying components as sources of error propagation (i.e., a component internal failure results in a propagation)
- *error paths* and *sinks* for specifying how components respond to incoming propagations (i.e., whether a particular error propagation is passed on as is to other components, is propagated to other components as a different error type, or is contained by the component)
- *propagation paths*, determined by the logical and physical connectivity in the architecture, the deployment of software on hardware, and user-defined propagation paths not recorded in the AADL core model
- probability properties associated with the occurrence of error propagations, sources, paths, and sinks

Figure 24 illustrates these concepts. Figure 27 shows a textual specification of a position sensor as an AADL device complete with a specification of its outgoing error propagation acting as an error source. It also includes descriptive information about it as a hazard source.

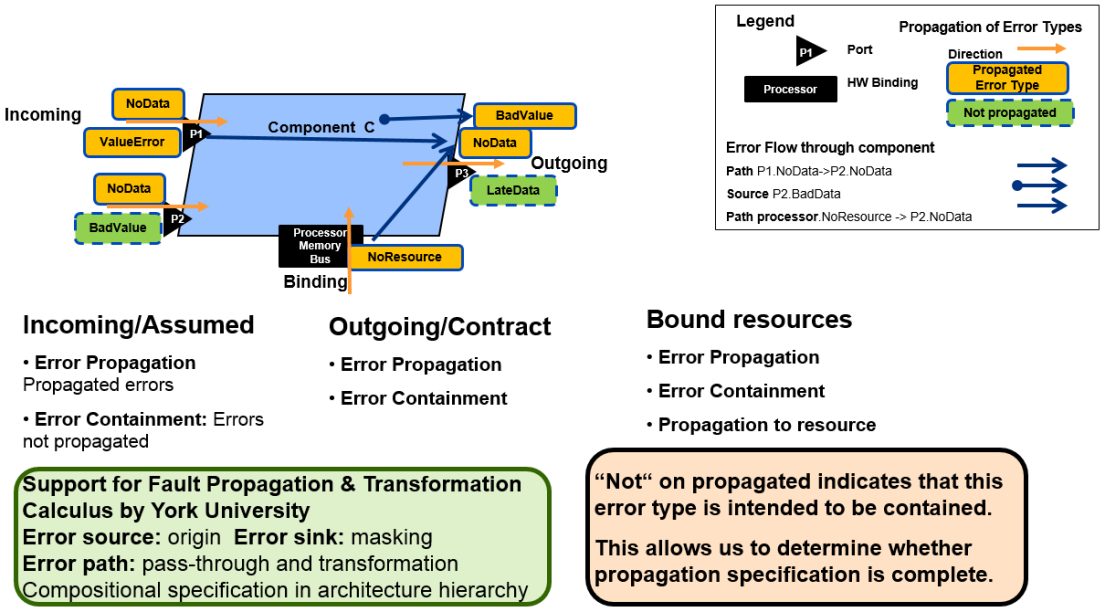

Figure 24: Error Propagations and Containment Specifications as Contracts

Figure 25 illustrates error types associated with outgoing and incoming ports to indicate error propagations, shown as rectangles of different colors. The propagation path between components follows the port connection between the components A and B. Component A is shown to be a source of error propagations of a specific type caused by a component A failure of a particular type (shown as colored oval). Component A also passes on incoming errors from its in port through its out port.

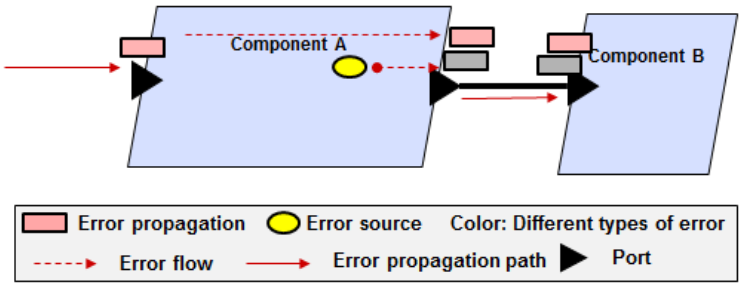

Figure 25: Error Propagation Between Components

3.3 EMV2 Support for Hazard Specification

The Error Model Annex includes a set of properties that are defined in the property set EMV2. One such property is *Hazards*, which has a record structure to capture all relevant aspects of a hazard description in the context of an FHA. It allow modelers to provide descriptive hazard information to the model. Figure 26 shows the elements of this record structure.

```
Hazards: list of record
   (
    CrossReference : aadlstring;    -- cross reference to an external document
    HazardTitle : aadlstring;       -- short descriptive phrase for hazard
    Description : aadlstring;       -- description of the hazard (same as hazardtitle)
    Failure : aadlstring;           -- system deviation resulting in failure effect
    FailureEffect : aadlstring;     -- description of the effect of a failure (mode)
    Phases : list of aadlstring;    -- operational phases in which the hazard is relevant
    Environment : aadlstring;       -- description of operational environment
    Mishap : aadlstring;            -- description of event (series) resulting in
                                    -- unintentional death, etc.(MILSTD882)
    FailureCondition : aadlstring; -- description of event (series) resulting in
                                    -- unintentional death, etc.(ARP4761)
    Risk : aadlstring;              -- description of risk. Risk is characterized by
                                    -- severity, likelihood, and occurrence probability
    Severity : EMV2::SeverityRange ;              -- actual risk as severity
    Likelihood : EMV2::LikelihoodLabels;          -- actual risk as likelihood/probability
    Probability: EMV2::ProbabilityRange;             -- probability of a hazard
    TargetSeverity : EMV2::SeverityRange;         -- acceptable risk as severity
    TargetLikelihood : EMV2::LikelihoodLabels;    -- acceptable risk as likelihood/prob
    DevelopmentAssuranceLevel : EMV2::DALLabels; -- level of rigor in development
                                                  -- assurance (ARP4761)
    VerificationMethod : aadlstring; -- verification method to address the hazard
    SafetyReport : aadlstring;       -- analysis/assessment of hazard
    Comment : aadlstring;
    )
```

Figure 26: Elements of an EMV2 Hazard Description

The modeler examines each component type of interest and determines whether it is a potential hazard source. They do that by assigning a set of hazard property values to error sources, outgoing error propagations of components. These assignments can be specific to a particular error type. Severity and Likelihood values can be assigned as part of the Hazards record, or they can be assigned as separate property values. Those values are used if the Severity or Likelihood value in the Hazard record is not set.

Figure 27 illustrates an example hazard specification that is associated with a system component (*PositionSensor*). The Hazard property is associated with the *Failed* state of an error source. Such hazard specifications are characterized with severity and criticality.

```
device PositionSensor
features
  PositionReading: out data port ;
flows
  f1: flow source PositionReading {Latency => 2 ms .. 3 ms;};
annex EMV2 {**
use types ErrorLibrary;
use behavior ErrorModelLibrary::Simple;
error propagations
  PositionReading: out propagation {ServiceOmission, ItemOmission, ValueError};
flows
  ef1:error source PositionReading when Failed;
end propagations;
properties
  EMV2::hazards =>
    ([ crossreference => "1.1.1";
```

```
        failure => "Loss of sensor readings";
        phases => ("all");
        severity => 1;
        likelihood => C;
        description => "No position readings due to sensor failure";
        comment => "Becomes major hazard, if no redundant sensor";
        ])
    applies to ef1.Failed;
**};
end PositionSensor;
```

Figure 27: Example Hazard Specification

Tailored versions of the Hazards property are defined in the property set ARP 4761 and MILSTD882. They use labels consistent with the respective standard. For a full discussion of the different forms of safety analysis, including FHA modeling with AADL and EMV2 see Delange [Delange 2014].

Once all relevant system components (i.e., their component type declarations) have been annotated with EMV2 error source or propagation declarations and hazard information, we create an AADL instance of the ASSA and invoke the FHA tool in OSATE. This produces a tabular functional hazard assessment report [Delange 2014].

4 Safety Analysis of ASSA

In this section, we apply the ALSA process to the ASSA system—the target of the JCA Demo ACVIP shadow project. The ASSA is the combined integration of the DCFM software component with the MIS system. The first step is to understand the requirements from a stakeholder perspective. This is usually done by examining a set of mission scenarios in the context of a Concept of Operation (CONOP) description.

In our case study of the ACVIP shadow project, we are dealing with an aircraft in an operational mission context. The focus is not on hazards in the flying aircraft itself, but on hazardous in-flight conditions as the aircraft interacts with entities in its operational environment that can potentially lead to loss of aircraft. In other words, we are identifying accident categories and focus on those that address interactions between the aircraft and its environment.

Figure 28 illustrates the aircraft and entities in its operational environment whose interactions can potentially lead to an accident in the form of aircraft loss. In our case study the stakeholder requirement document provided insight into the entities to be considered. They are captured in the AADL model as a set of abstract components and placed into a type hierarchy, similar to representing them as a Unified Modeling Language (UML) class diagram. In Figure 28, we have only elaborated the type hierarchy for threats.

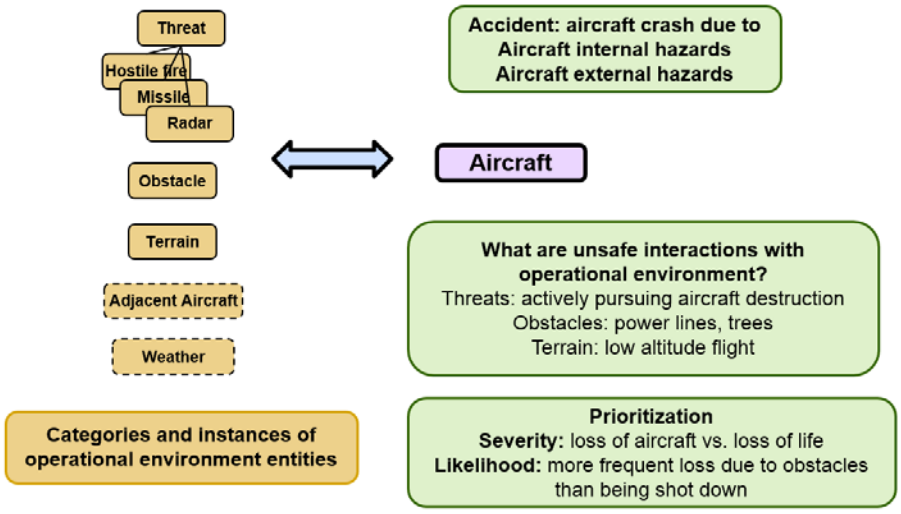

Figure 28: Aircraft in Its Operational Environment

As noted in the SEI special report *Requirement and Architecture Specification of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System*, several types of obstacles are mentioned in the textual stakeholder requirement document—but their relationship was ambiguous [Feiler 2015b]. Another interesting observation was that although obstacles were indicated as leading to a higher loss of aircraft, obstacle tracking was not included in the textual system requirement specification document.

A common way of viewing a system in its operational context is via *Monitored* and *Controlled Variables*. This approach—documented in the FAA Requirement Engineering Management Handbook [FAA 2009]—has its roots with Parnas [Parnas 1991]. These variables represent a state that can be used to characterize unsafe system conditions and interactions. To operationalize this view we introduce sensors and actuators to represent the monitored and controlled variables. Figure 29 illustrates this. Note that some systems are under our control, while other systems we can only observe.

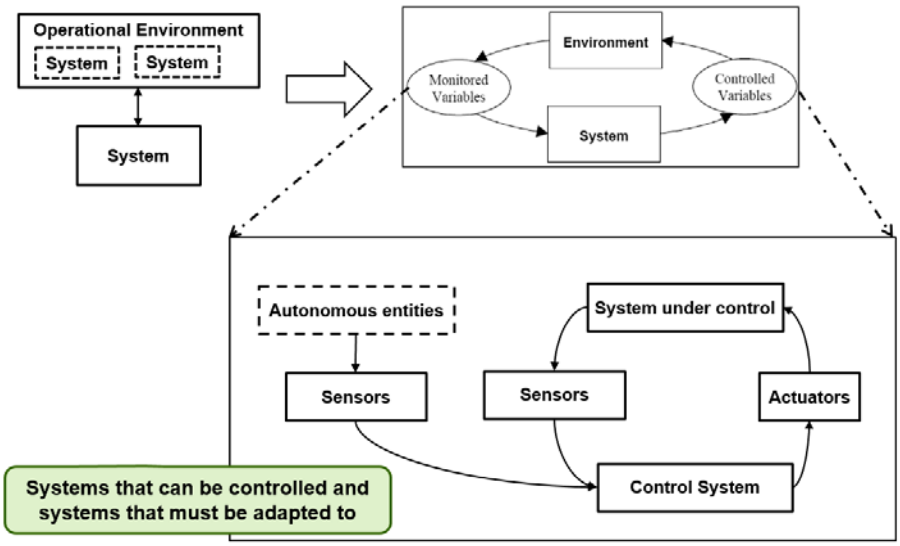

Figure 29: From CONOP to a Control System Architecture View

Figure 30 illustrates this control system view for our case study. The pilot acts as the control system for the aircraft—utilizing appropriate sensors for input and actuators to issue commands to the aircraft. The ASSA plays the role of a sensor with respect to entities in the operational environment that have been identified in the previous phase as potentially leading to accidents. In other words, the ASSA is an intelligent system that makes the pilot aware of hazardous situations that affect aircraft survivability.

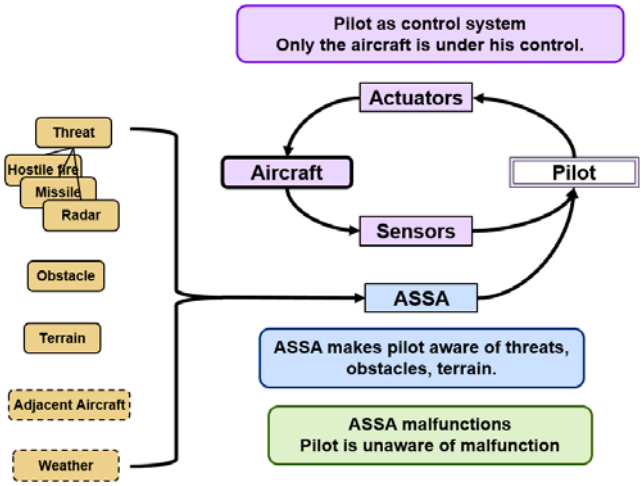

Figure 30: ASSA as Sensor in a Control System

To identify the hazards ASSA poses to the pilot, we use the EMV2 fault ontology as a check list. Figure 31 shows the resulting identified hazards.

Hazards in interaction with pilot due to ASSA malfunction

- Missing (false negative) SA information
- Erroneous (false positive) SA information
- Incorrect SA information (incorrect position)
- Untimely SA information (late arrival: perceived as incorrect time sensitive data)
- Time-inconsistent SA information (asymmetric position timing error)

Category specific sensing hazards

- Different thresholds for threats, obstacles, terrain
- Failure to sense specific categories

Figure 31: Potential ASSA Hazards

The process involves interaction between a subject matter expert and a safety analyst to identify the actual hazard in the application domain of situational awareness (SA) systems. It utilizes the functional architecture model of ASSA as context (shown in Figure 32). This architecture has been captured as a reference architecture for SA systems in AADL [Feiler 2015b]. We annotate this AADL model with fault information through EMV2 declarations.

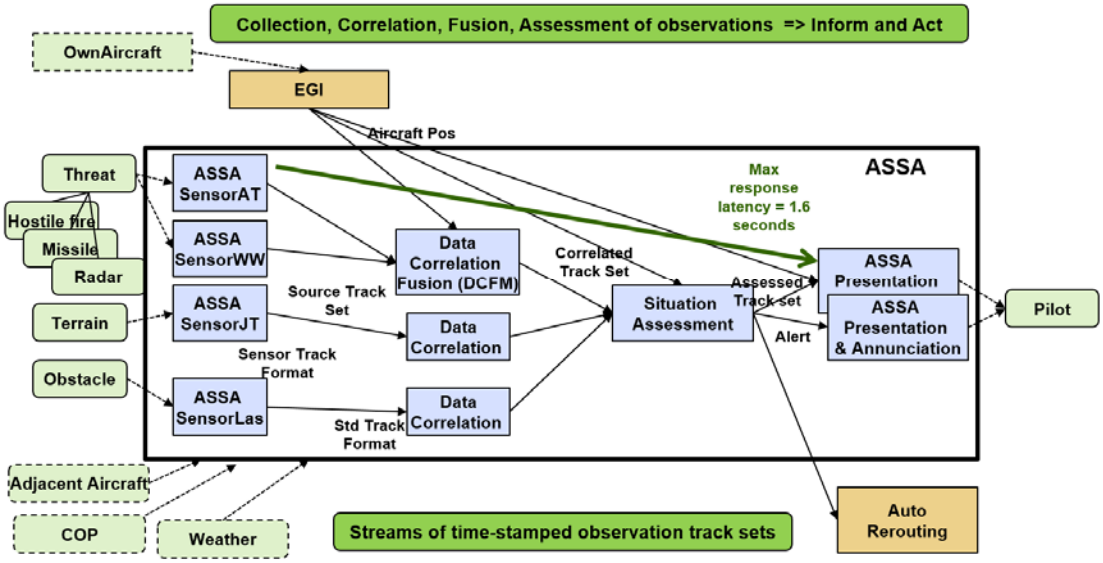

Figure 32: ASSA System Functional Architecture

First, we consider omission. This may take the form of service omission (i.e., the ASSA failing to operate), or item omission (i.e., the ASSA failing to inform the pilot of a present threat or obstacle). To the pilot this appears as missing SA information. As a result of this hazard, the pilot may not take evasive action to avoid the threat of obstacle.

Second, we consider commission. This takes the form of informing the pilot of threats or obstacles that do not exist (i.e., presenting the pilot with erroneous SA information). This presents a

risk in that the pilot may take an action to avoid a non-existent obstacle and unnecessarily expose the aircraft to enemy fire.

Third, we consider value errors. These take the form of showing an incorrectly calculated position relative to the own aircraft position and incorrectly determining the threshold for warning and annunciation.

Fourth, we consider timing errors. These take the form of taking longer than expected in performing the processing and providing the information to the pilot (i.e., the end-to-end latency in the information flow from the sensors to the pilot display). Since we are dealing with time-sensitive information, the result is for the pilot to be informed too late and with inaccurate information.

Finally, we consider replication errors. In the case of the ASSA, there is the potential for two information paths involving own aircraft position, one passing this position information directly from the embedded global positioning satellite/inertial navigation system (EGI) to the pilot display and the other feeding the position through the ASSA to compute aircraft-relative position information of entities in the operational environment. The effect is an asymmetric timing error on the pilot display, which translates into inaccurate information.

As we have different categories of threats, obstacles, and terrain, there may be different thresholds for warning and annunciation. There is also the issue of partial failure of the ASSA (i.e., sensor failure for certain categories of these entities in the operational environment that the pilot should be aware of).

We record this information in two steps.

First, we define a set of error types that reflect the types of fault propagation in a domain-specific set of guide words. We do this by defining an error type library called *ExceptionalConditionTypes* with error types as extensions of the predeclared EMV2 error types (fault ontology), as aliases (*renames*) of existing types, or as new error types as shown in Figure 33.

```
package ExceptionalConditionTypes
public
annex EMV2 {**
 error types
  ASSALoss: type extends ErrorLibrary::ServiceOmission;
  InvalidASSAData: type;
  DegradedASSAData : type extends InvalidASSAData;
  StaleASSAData : type extends InvalidASSAData;
  FalsePositiveASSAData: type;
  FalseNegativeASSAData: type extends ErrorLibrary::ItemOmission;
  OutOfRangeASSAData renames type ErrorLibrary::OutOfRange;
  UntimelyASSAData: type extends ErrorLibrary::LateDelivery;
  TimeSkewedASSADataAircraftPosition: type extends ErrorLibrary::AsymmetricTiming;
  InaccurateASSAData: type set {OutOfRangeASSAData, TimeSkewedASSADataAircraftPosi-
tion};
  ASSAFaults: type set {FalsePositiveASSAData,FalseNegativeASSAData, Inaccurate-
ASSAData };
  -- consequent accident
  AircraftLoss: type extends ErrorLibrary::ServiceOmission;
```

```
  --Track error types
  --sequence errors
  ObservationsOutofOrder: type extends ErrorLibrary::OutOfOrder;
  ObservationMissing: type extends ErrorLibrary::ItemOmission;
  --Geographic error types
  --Inaccurate and imprecise
  ObservationsImprecise: type extends ErrorLibrary::ValueError;
  ObservationsInaccurate: type extends ErrorLibrary::ValueError;
  --Position errors
  AirspeedValueError: type extends ErrorLibrary::ValueError;
  AltitudeValueError: type extends ErrorLibrary::ValueError;
  LongitudeValueError: type extends ErrorLibrary::ValueError;
  LatitudeValueError: type extends ErrorLibrary::ValueError;
  -- track set related errors
  TrackSetIntegrity renames type ErrorLibrary::ValueError;
  TrackOmission renames type ErrorLibrary::ItemOmission;
  TrackValueError : type extends ErrorLibrary::ValueError;
  TrackComputationError : type extends ErrorLibrary::ValueError;
 end types;
**};
end ExceptionalConditionTypes;
```

Figure 33: ASSA-Specific Error Types

Second, we use the guide words to perform the safety analysis. We annotate the interaction points (ports, feature groups, etc.) of ASSA with error propagation declarations that reference the error types we just defined. Figure 34 shows the outgoing propagations to the pilot. It also shows that we do not expect the *OwnAircraftPosition* to be out of range. Finally, it shows that missing own aircraft position information is mapped into *FalseNegativeASSAData* to indicate that an entity may falsely be considered absent.

```
    annex emv2 {**
        use types ErrorLibrary, ExceptionalConditionTypes;
        error propagations
        SSSAirCrewPresentation: out propagation {FalsePositiveASSAData, FalseNegativeASSAData};
        SSSAirCrewPresentation.AircrewTerrainLocationAwareness: out propagation
             {FalsePositiveASSAData, FalseNegativeASSAData};
        OwnAircraftPosition: not in propagation {OutofRange};
        OwnAircraftPosition: in propagation {ItemOmission};
        flows
            passthrough: error path OwnAircraftPosition{ItemOmission}
             -> SSSAirCrewPresentation{FalseNegativeASSAData};
        end propagations;
        properties
        EMV2::Severity => 2 applies to SSSAirCrewPresentation.AircrewTerrainLocationAwareness;
    **} ;
end ASSASystem;
```

Figure 34: EMV2 Annotation of the ASSA System Interface

Once we have elaborated the functional and physical architecture of ASSA (see also Feiler 2015b), we examine each ASSA component from a safety analysis perspective, identifying potential hazard contributors. The steps are outlined in Figure 35.

For each ASSA component

- Identify outgoing error propagations as error sources
  - The component itself is the hazard contributor
- Identify outgoing error containments
  - Promises/guarantees of not propagating certain errors
- Identify incoming error propagations and paths
  - Assumptions about masking (sink), pass through/conversion to different type (path)
  - Incoming through ports as well as deployment/resource bindings
- Identify incoming error containment as assumption

Fault impact analysis

- Mismatched propagation assumptions (unhandled faults)
- Failure modes and effects analysis

Figure 35: Identifying Hazard Contributors

In the first step, we identify potential hazard sources for each of the components that interact with the ASSA. In our example, we annotate the EGI unit that provides aircraft position information. We then instantiate an AADL model that includes an instance of ASSA, EGI, and the pilot and invoke the FHA tool in OSATE. This results in an FHA report as illustrated in Figure 36.

Componen	Error	Hazard Description	Crossrefer	Functional	Operation	Severity	Likelihood
egi1	"ItemOmission on airCraftPosition"	"Lack of providing aircr	"N/A"	"Lack of Ai	"all"	3	C
assa	"FalsePositiveASSAData on SSSAirCrewPresentation"	"Reporting of non-exist	"N/A"	"Non-Exist	"all"		
assa	"FalseNegativeASSAData on SSSAirCrewPresentation"	"Failed reporting of thr	"N/A"	"Failed Re	"all"		
assa	"FalsePositiveASSAData on SSSAirCrewPresentation"	"Inaccurate and untime	"N/A"	"Inaccurat	"all"	2	
assa	"FalseNegativeASSAData on SSSAirCrewPresentation"	"Inaccurate and untime	"N/A"	"Inaccurat	"all"	2	
assa	"InaccurateASSAData on SSSAirCrewPresentation"	"Inaccurate and untime	"N/A"	"Inaccurat	"all"	2	

Figure 36: A Sample FHA Report

Note that some of these hazard contributors are design defects that can be eliminated during design. In EMV2 we have the ability to tag different error sources as to whether they are to be considered design errors that can be eliminated by design revisions or operational exceptional conditions that impose requirements on a safety system. Figure 37 shows some candidates for elimination by design.

Data exchange inconsistency

- Measurement units, base type representation, standard domain representation
  - Functional integration consistency checking of data model
- Elimination
  - Matching data interchange assumptions and guarantees

Time inconsistent aircraft position related data

- Multiple paths with different latencies
  - Identification through analysis
  - Impact of partition design decisions
- Elimination
  - Redesign to achieve same path latencies
  - Accommodate for discrepancy in data correlation projections

Elimination indicated by error
EMV2 containment declaration
Evidence by verification activities

Incorrect assessment thresholds

- Operational scenario simulation and human factor study
  - Category specific thresholds

Figure 37: Error Mitigation by Design

In a second step, we revisit each component type to identify any incoming error propagations from other components and how they are addressed (i.e., whether they are masked [error sink] or passed through to other component [error path to an outgoing error propagation]).

Once we have specified all outgoing and incoming error propagations, the EMV2 consistency analysis identifies mismatched assumptions about error propagation along each of the connections between components. For example, the consistency analysis identifies unhandled faults where an outgoing propagation from one system component is not expected as incoming error propagation by a receiving system component. Figure 38 shows such an unhandled fault identification by ASSA when its incoming error propagation does not include value errors.

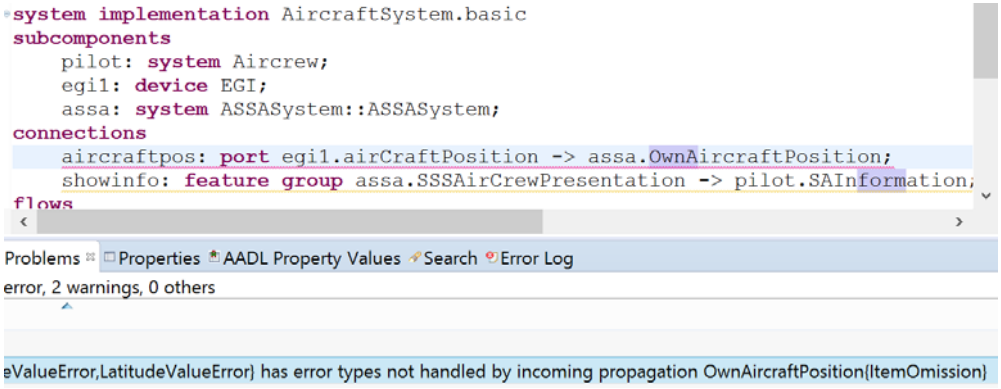

Figure 38: Unhandled Fault Identification

Similarly, a fault impact analysis uses the same to trace the impact of error sources through the system by following a propagation path identified by connections and error flows. In our example, it traces hazard contributors to the ASSA hazards and to resulting incidents. Figure 39 shows a fault impact analysis report for ASSA in its operational context that includes the EGI before the ASSA model has been expanded into its subsystems. The report shows the component and the initial failure mode. The error type shown is that of the error source declaration. If the error type is not present in the error source, the error type of the outgoing error propagation is shown. The first-level effect column shows the outgoing effect, identifying the outgoing error propagation type and the destination. The entry then repeats the failure mode of the affected component and its propagation as a next-level effect. This generic fault impact report can be adapted to specific FMEA formats.

Fault Impact of System Internal Error Sources

Component	Initial Failure Mode	1st Level Effect	Failure Mode	second Level Effect
egi1	{ItemOmission}	{ItemOmission} airCraftPosition -> assa:OwnAircraftPositior	assa {ItemOmission}	{FalseNegativeASSAData} SSSAirCrewPresentati
egi1	{AirspeedValueError}	{AirspeedValueError} airCraftPosition -> assa:OwnAircraftPc	assa {AirspeedValueError}	{InaccurateASSAData} SSSAirCrewPresentation
egi1	{AltitudeValueError}	{AltitudeValueError} airCraftPosition -> assa:OwnAircraftPo	assa {AltitudeValueError}	{InaccurateASSAData} SSSAirCrewPresentation
egi1	{LongitudeValueError}	{LongitudeValueError} airCraftPosition -> assa:OwnAircraftF	assa {LongitudeValueError}	{InaccurateASSAData} SSSAirCrewPresentation
egi1	{LatitudeValueError}	{LatitudeValueError} airCraftPosition -> assa:OwnAircraftPo	assa {LatitudeValueError}	{InaccurateASSAData} SSSAirCrewPresentation
assa	{InaccurateASSAData}	{InaccurateASSAData} SSSAirCrewPresentation -> [No Outgoing Conn]		
assa	{SensorLoss}	{FalsePositiveASSAData} SSSAirCrewPresentation -> [No Outgoing Conn]		
assa	{FalseNegativeASSAData}	{FalseNegativeASSAData} SSSAirCrewPresentation -> [No Outgoing Conn]		

Figure 39: Fault Impact Analysis Report for ASSA

Once we have elaborated the functional and system architecture of ASSA (see also the SEI special report CMU/SEI-2015-SR-030 [Feiler 2015b]), we examine each ASSA component from a safety analysis perspective, identifying potential hazard contributors. They are recorded as EMV2 annotations for each of the ASSA. Figure 40 shows a sampling of potential hazard contributors by the functional architecture of ASSA. When rerunning the FHA tool these potential hazards will be included in the report.

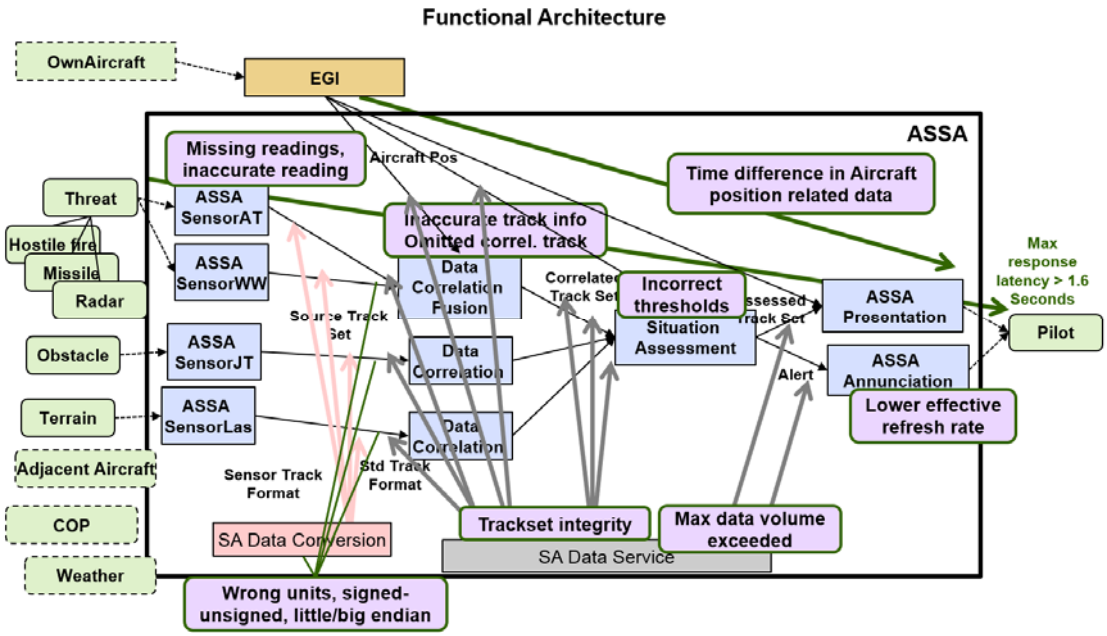

Figure 40: Sampling of Functional Architecture Hazard Contributors

Separate from the functional architecture, we examine the hardware platform system architecture of ASSA for hazard contributors. It consists of the sensors, processor, display, and network components. All of them are dependent on external power supply. This architecture and its hazard contributors are shown in Figure 41. In the example, we have focused on component failures which manifest themselves as service omission error propagations. In the case of the general processing unit (GPU), data corruption in memory is being considered. In the case of the network labelled Ethernet, we consider dropped packets (i.e., item omission).

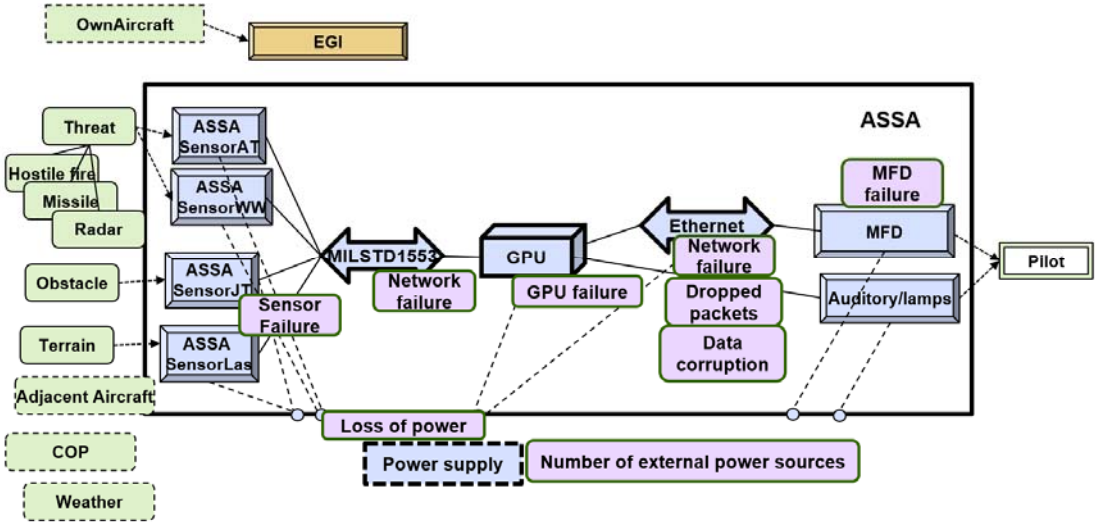

Figure 41: Hazard Contributions by Physical Architecture

Again, once the AADL model of the hardware platform is annotated with incoming error propagations and error paths, we perform a fault impact analysis to get a report of how failure of a hardware platform component affects other hardware platform components.

Finally, we define an ASSA system configuration that includes the hardware platform and add a binding specification of the functional architecture to elements of the hardware platform using AADL binding properties. We also revise the EMV2 annotations of functional architecture components being bound to the platform and platform components to identify potential propagations of error types between the functional architecture and platform due to the binding. After we reran the fault impact analysis, the report now includes the effects of failures in the platform propagating to the ASSA system.

5 ASSA Health Monitoring System Requirements

Once the ASSA hazards have been identified, we can define a set of derived requirements for a Health Monitoring System for the ASSA. The original textual system requirement document included requirements for a health monitor. Its requirement statements focused primarily on assessing the health of the sensors at startup time. They stated that as long as one sensor was operational the system is to enter operational mode. The requirement document did not cover monitoring during operation and the kind of health information to report to the pilot (e.g., which subset of sensors is operational).

In the context of the ALSA process, we can use the results of the safety analysis to determine which ASSA components are to be monitored during operation (i.e., whose conditions have not been eliminated during design and can actually be detected). Figure 42 shows that in our example the health monitor should track whether there are missing sensor readings and whether the maximum data volume is exceeded (i.e., available storage is exhausted).

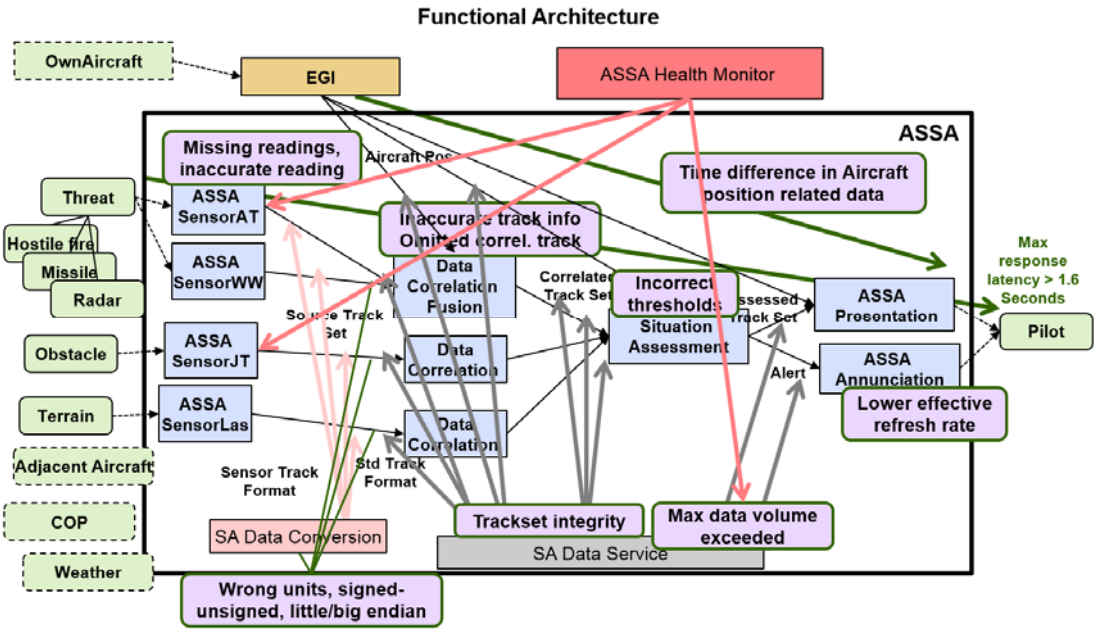

Figure 42: Identification of Functional Components to Be Monitored

Figure 43 illustrates the physical components to be monitored for failure (omission) and, in the case of the Ethernet network, also for dropped packets and data corruption.

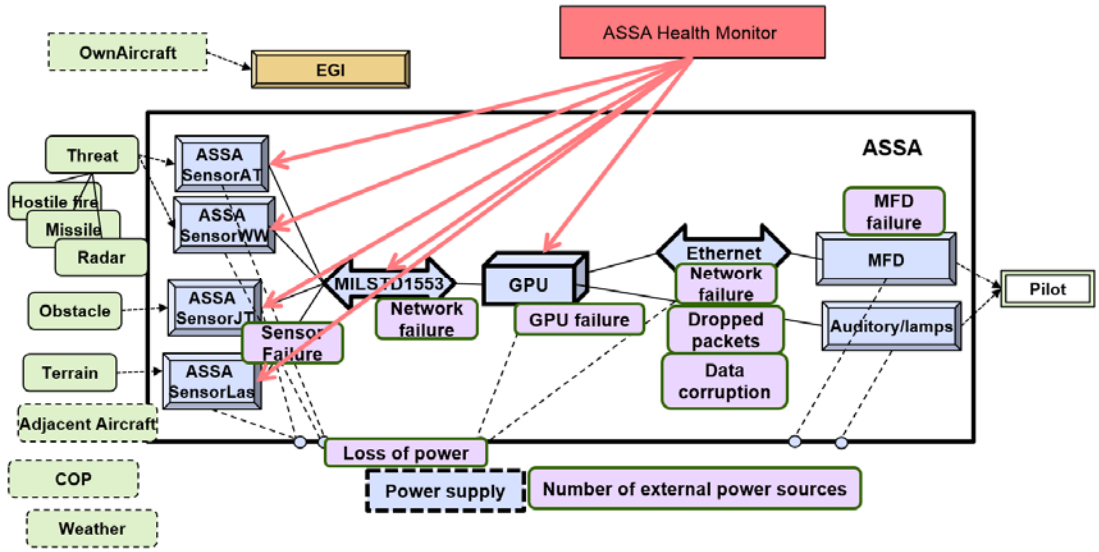

Figure 43: Identification of Physical Components to Be Monitored

Finally, we apply the safety analysis process to the Health Monitoring System itself. This is illustrated in Figure 44. The health monitor is a piece of software that must execute on a processor. If it is bound to the same processor as the ASSA functions it is responsible for monitoring, then a processor failure cannot be detected and reported by the health monitor. This is determined by a fault impact analysis whose impact trace shows that a GPU failure propagates to both the ASSA functions and the health monitor functions.

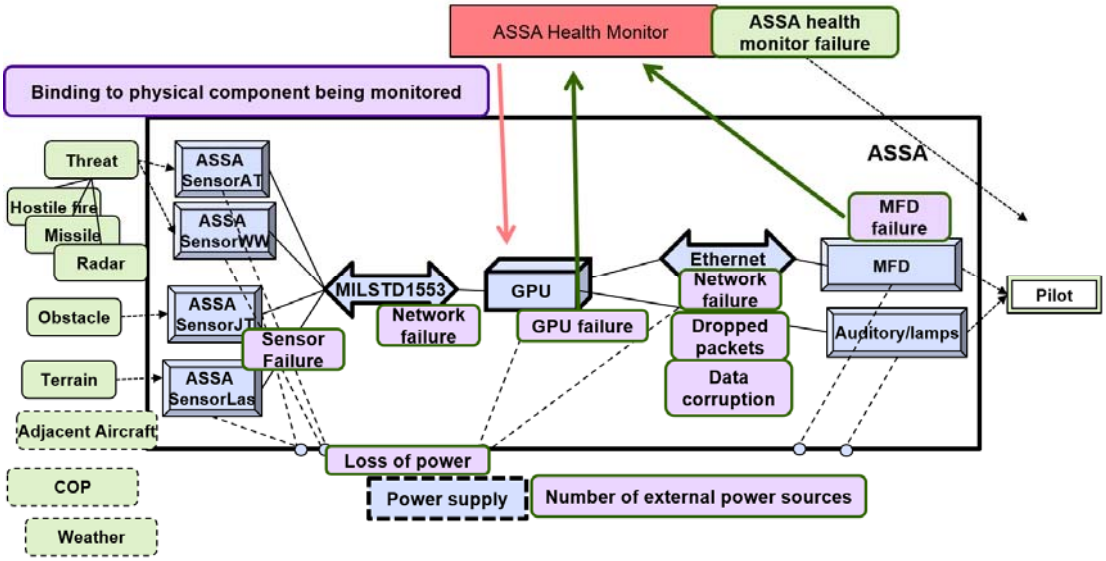

Figure 44: Safety Analysis of Health Monitoring System

6 Summary and Conclusion

The purpose of the JCA Demo ACVIP shadow project was to demonstrate the value of using ACVIP technology, in particular the use of architecture models expressed in the SAE AADL standard, in discovering potential system integration problems early in the development process. The SEI team demonstrated two ACVIP processes in this project, one for Architecture-Led Requirements Specification (ALRS) and another for the Architecture-Led Safety Analysis (ALSA).

The SEI special report *Requirement and Architecture Specification of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System* summarizes the ALRS process that was used to capture requirements from existing requirement documents in an AADL model annotated with requirement specification declarations [Feiler 2015b]. In that process, we analyzed the resultant AADL model for potential system integration issues, which we reported in SEI special report *Potential System Integration Issues in the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System* [Feiler 2015a].

In this report, we described an ALSA process that leverages and integrates with the ALRS process. We used the EMV2 standard to annotate the AADL with fault information. This allowed us to automate various safety-related analyses outlined in SAE ARP 4761. The fault propagation ontology of EMV2 allowed us to derive domain-specific guide words to identify the hazards of an ASSA system and various hazard contributors. We identified several types of hazards: false negative, false positive, incorrect, untimely, and time-inconsistent SA information. We showed how such safety analysis can be repeated for several layers of a system architecture by performing it on ASSA in its operational context and then repeating it for the ASSA subcomponents.

Automation of safety analysis allows us to make architectural changes or enrich the fault annotations and repeat the safety analysis at little incremental cost. This makes safety analysis of subsystems affordable.

We have showed how safety analysis can be applied to functional and system architectures separately and then repeated after the functional architecture is mapped to the system architecture. As the ASSA is largely a software-based system, we showed that this architecture-led safety analysis allows us to consider software faults as a major hazard contributor and understand the impact of such fault occurrences.

We illustrated functional hazard assessment and fault impact analysis. The same models can be the basis for common cause analysis, fault tree analysis, and, after annotation with fault occurrence, distributions for probabilistic safety analyses. We then used the safety analysis results to identify those hazard contributors that can be eliminated by design and a set of requirements for a health monitoring system for ASSA, whose responsibility is to inform the pilot of ASSA malfunctions. In that context, we identified that the scope of responsibility for the health monitor was not well defined in the original requirement document. Finally, we subjected the health monitoring system itself to a hazard analysis and identified a potential common cause hazard if the ASSA functional software and health monitoring software are allocated to the same processor.

Appendix Acronym List

AADL	Architecture Analysis & Design Language
ACVIP	Architecture-Centric Virtual Integration Process
ALRS	architecture-led requirement specification
ALSA	architecture-led safety analysis
AMRDEC	Aviation and Missile Research, Development, and Engineering Center
ASSA	Aircraft Survivability Situation Awareness
ATAM	Architecture Tradeoff Analysis Method
BAA	Broad Agency Announcement
CPRET	Constraints, Products, Resources, Elements, and Transformation
CRC	cyclic-redundancy check
DCFM	Data Correlation and Fusion Manager
EGI	embedded GPS/INS
EMV2	Error Model Annex V2
FHA	Functional Hazard Assessment
FMEA	Failure Mode and Effect Analysis
HAZOP	hazard and operability
HMS	Health Monitoring System
INS	Inertial Navigation System
JCA	Joint Common Architecture
JMR	Joint Multi-Role
MIS	Modular Integrated Survivability
NACK	negative acknowledgment
NM	nautical miles
OSATE	Open Source AADL Tool Environment
SA	situational awareness
SADS	situational awareness data service
SEBoK	Software Engineering Body of Knowledge
SEI	Software Engineering Institute
SSS	System Segment Specification
STAMP	System Theoretic Accident Model Process
STPA	System Theoretic Process Analysis
UML	Unified Modeling Language

References

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

[ARP 4754A]
SAE Aerospace Recommended Practice, *Guidelines for Development of Civil Aircraft and Systems*, 2010-12, SAE ARP 4754A. 2010.

[ARP 4761]
SAE Aerospace Recommended Practice, *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, 1996-12, SAE ARP 4761. 1996.

[Delange 2014]
Delange, Julien et al. *AADL Fault Modeling and Analysis Within an ARP4761 Safety Assessment.* CMU/SEI-2014-TR-020. Software Engineering Institute, Carnegie Mellon University, 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=311884

[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. June 2012. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=10115

[FAA 2009]
Federal Aviation Administration. *Requirements Engineering Management Handbook* DOT/FAA/AR-08/32. 2009. http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/AR-08-32.pdf

[Feiler 2009]
Feiler, Peter H. Challenges in Validating Safety-Critical Embedded Systems. *Proceedings of SAE International AeroTech Congress*. November 2009.

[Feiler 2015a]
Feiler, Peter H. *Potential System Integration Issues in the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System*. CMU/SEI-2015-SR-031. Software Engineering Institute, Carnegie Mellon University. 2015. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=447176

[Feiler 2015b]
Feiler, Peter H. & Hudak, John. *Requirements and Architecture Specification of 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=447184

[Goodin 2015]
Goodin, Dan. Boeing 787 Dreamliners Contain a Potentially Catastrophic Software Bug. *Ars Technica Technology Lab*. May 1, 2015. http://arstechnica.com/information-technology/2015/05/boeing-787-dreamliners-contain-a-potentially-catastrophic-software-bug/

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

[Parnas 1991]
Parnas, D. & Madey, J. *Functional Documentation for Computer Systems Engineering (Version 2)*, Technical Report CRL 237. McMaster University. September 1991.

[Paige 2009]
Paige, Richard F. et al. FPTC: Automated Safety Analysis for Domain-Specific Languages. In Models in Software Engineering. *Lecture Notes In Computer Science*, Volume 5421. 2009. Pages 229-242.

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

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

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
Architecture-Led Safety Analysis 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-032

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) team was involved in an Architecture-Centric Virtual Integration Process shadow project for the U.S. Army's Aviation and Missile Research, Development, and Engineering Center (AMRDEC) Science & Technology Joint Multi-Role (JMR) vertical lift program on the Joint Common Architecture (JCA) Demonstration. The JCA Demo used the Modular Integrated Survivability (MIS) system. The MIS project provided a Situational Awareness Data Manager service that was integrated with Data Correlation and Fusion Manager (DCFM). This report summarizes the approach taken in the architecture-led safety analysis of what will be referred to as the JMR Aircraft Survivability Situation Awareness (ASSA) system. The ASSA system was the focus of the Phase 2 MIS project, in which an AMRDEC team developed support services for ASSA and contractors provided a DCFM component. These components were implemented to conform to the Future Airborne Capability Environment (FACE™) Standard specification for portability and integrated on two hardware platforms. By taking an architecture-led approach to safety analysis, the SEI team demonstrated the use of Architecture Analysis and Design Language and the Error Model V2 Annex standard in performing safety analysis of an embedded software system.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Architecture, safety, AADL, ACVIP	47

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