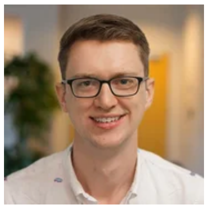
Search the blog

Software Engineering Institute

SEI Blog

Home > Publications > Blog > The AADL Error Library: 4 Families of Syste...

The AADL Error Library: 4 Families of System Errors



SAM PROCTER

MAY 20, 2019

PUBLISHED IN

Software Architecture

CITE

Get Citation

TAGS

Software Architecture | Architecture Analysis and Design Language (AADL)

Classifying the way that things can go wrong in a component-based system is a hard challenge since components--and the systems that rely on them--can fail in myriad, unpredictable ways. It is nonetheless a challenge that should be addressed because component-based, software-driven systems are increasingly used for safety-critical applications. Unfortunately, many well-established classifications and taxonomies of system errors are not what we would term *operationalized* (i.e., directly usable in modern, model-based system engineering efforts). Instead, they are specified and described in natural language instead of any formal or semiformal specification language.

In this blog post, which is adapted from a recently published paper, we present the Architecture Analysis and Design Language's (AADL) EMV2 Error Library, which is an established taxonomy that draws on a broad range of previous work in classifying system errors. The second version of AADL's error modeling annex (EMV2) Error Library is now part of an international standard that has been used in a range of systems and domains.

The AADL EMV2 Error Library is deeply integrated in AADL, a rich, semi-formal embedded system modeling language. The errors included in the library have formalized semantics and the library is designed to be easily extended by system developers to become domain- or system-specific. Specifically, the EMV2 Annex provides:

- An ontology of system errors (i.e., the Error Library) that is embeddable into system architecture models that have been specified in AADL. This ontology relates error types to one another through type extension, which enables modeling different layers of abstraction. The types are also instantiable into tokens, which flow through Petri-net-like specifications of component error behavior.
- Formal specifications of the semantics of the error types in the library. These are not included in the blog post, but there are a few examples included in the paper.

The Importance of Effects-Based Reasoning

One of the key aspects of the AADL EMV2 taxonomy is its enforcement of local, effects-based error classification. The importance of focusing on the effects of errors (rather than their causes) in system analysis has been discussed previously, including by Walter and Suri, who made it an important part of their *Customizable Fault/Error Model*. At a high level, the primary justifications for taking an effects-based approach include

- **Reducing Ambiguity**: Walter and Suri argue that cause-based classification methods, such as the Laprie taxonomy, can lead to the same fault being classified differently in different systems, but note that this is impossible with an effects-focus.
- **Reducing Analysis Space**: Procter et al. note that the set of causes is essentially unbounded, while observable fault effects can, at a given level of abstraction, be known statically--making analysis of larger systems more tractable.
- **Enabling Local Reasoning**: As the analysis space is reduced to a more manageable level, it is possible to gain a notion of completeness and minimality with some effect-classification taxonomies (eg, Dolev and Yao's). This reduction means that, to a limited extent, a component can be analyzed independently of other parts of the system of which it is a part.
- **Connecting to Compositional Reasoning**: There is an important connection between effects-based reasoning, assume/guarantee logics, and design by contract styles. Errors can also be conceptualized as violations of a component's assumptions. Similarly, the effects of those errors are typically violations of the component's guarantees.
- **Enabling Safety and Security Co-Analysis**: There is a growing recognition of the overlap between traditional safety and security concerns. An effects focus allows, in some cases, for the simultaneous co-analysis of safety and security, a topic that has been previously explored.

4 Families of Errors in the AADL EMV2 Error Library

This blog post isn't the appropriate place to fully describe the semantics of each error type; rather, full details are located in the AADL EMV2 standard. In this section of the post, we provide an overview of the families of error types used in the libraries.

Service Errors. We use the terms *sequence* and *service* to refer to bounded and unbounded (respectively) ordered collections of *service items* (i.e., messages, inputs, etc.), where accuracy and timeliness are required for correctness. The first family of error types, Service Errors, shown in Figure 1 below, contains errors in "the number of service items delivered by a service."

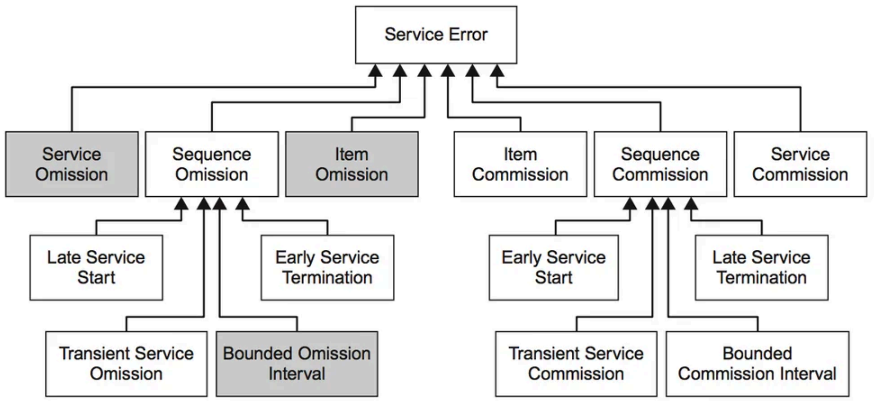


Figure 1. Hierarchy of service errors. Adapted from http://www.aadl.info/aadlinfosite/LinkedDocuments/AADLFeilerCACSD2006.pdf.

- **Commission and Omission.** Of the six error types that extend the top level ServiceError type, four deal directly with either unexpected items and services (commission) or missing ones (omission). If the errors are single events, the ItemCommission and ItemOmission types should be used; if it is an entire service that is in error, the Service counterparts should be used instead.
- **Sequences.** Errors that are more subtle than single service items or complete services are captured using the children of the SequenceOmission and SequenceCommission types. Some, such as early and late service start and termination, have an intuitive meaning. Two of the remaining four--TransientServiceOmission and TransientServiceCommission--are used for intermittent versions of their service-based counterparts. The final error types, BoundedOmissionInterval and BoundedCommissionInterval are used when service item errors occur more frequently than some specific bound. Note that if a sequence error persists longer than the system-specific *k* bound, it becomes a service error.

Value Errors. The second error family, shown in Figure 2 below, collects errors that represent incorrect values. The collection is split into three hierarchies: one dealing with items, one with sequences, and one with services.

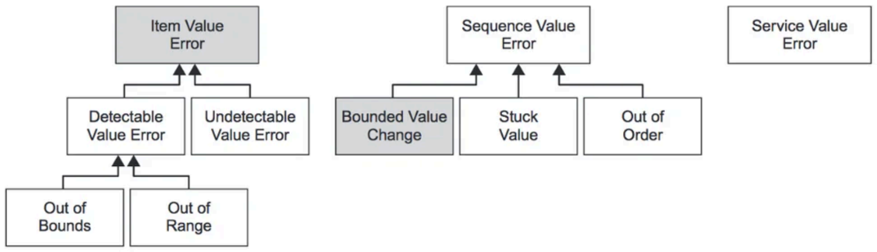


Figure 2. Hierarchy of value errors. Adapted from http://www.aadl.info/aadlinfosite/LinkedDocuments/AADLFeilerCACSD2006.pdf.

- **ItemValueError.** These errors deal with individual service items with incorrect values. The family is divided first by the detectability of the errors (i.e., if only an omniscient observer could detect them or if the system itself can as well). If they are detectable, errors can be classified as either out of range, which means they are outside of some domain-specific range (e.g., a percentage that is more than 100 or less than 0) or out of bounds, which means the value is unrepresentable in the expected type (e.g., a string is received instead of an integer).
- **SequenceValueError.** Some programs may be able to behave correctly when a small number of input values are incorrect, but longer sequences of erroneous values cannot be compensated for. This is further extended into out-of-order sequences and values that are Stuck, (i.e., repeating the same value). A final type, BoundedValueChange, signifies successive values that are both in range but are implausibly far apart (according to some user-specified boundary).
- **ServiceValueErrors.** Value errors with entire services signify that all service items have value errors.

Timing Errors. Timing errors, the third family of errors, can be challenging to model because two different notions of timing can be used. The first, *inter-arrival* time, specifies the length of time allowable between service items. The second, *clock time*, specifies delivery deadlines according to some more absolute notion of time (e.g., time-of-day, Unix time, time since system initialization, etc.). Both types of timing issues can be modeled using our library's timing error hierarchy, shown in Figure 3 below. Note that ItemTimingErrors can occur using either type of timing specification: they signify only that a single service item is either early or late according to some system-specific deadline.

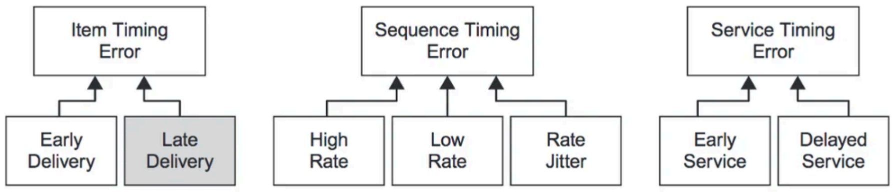


Figure 3. Hierarchy of timing errors. Adapted from http://www.aadl.info/aadlinfosite/LinkedDocuments/AADLFeilerCACSD2006.pdf.

- **SequenceTimingError.** This sub-family uses inter-arrival timing specifications. The more abstract SequenceTimingError can be refined into HighRate (violations of the minimum inter-arrival time), LowRate (violations of the maximum inter-arrival time), and RateJitter, which is a combination of the two.
- **ServiceTimingError.** This error sub-family uses clock timing specifications. The generic notion of ServiceTimingError is extended by both EarlyService, where service items arrive consistently early, and DelayedService, where they are late.

Replication Errors. The family of replication error types shown in Figure 4 below is used to model errors in replicated service items, which may come about as a result of various architectural mechanisms, (e.g., redundancy patterns, parallel execution, etc.) Work in this area was inspired in part by Walter and Suri's ideas on communication symmetry, which posited that otherwise undetectable errors could, in some systems, be detected if service items were replicated and used in multiple places.

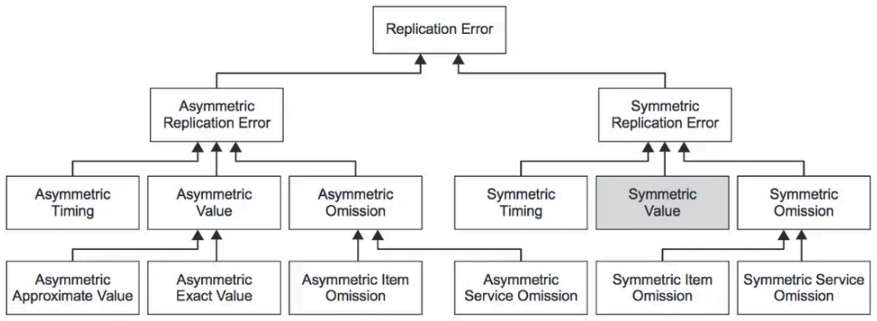


Figure 4. Hierarchy of replication errors. Adapted from http://www.aadl.info/aadlinfosite/LinkedDocuments/AADLFeilerCACSD2006.pdf.

In the error library, this gives rise to the family of replication error types. It combines the previous error families--service, value, and timing--with either symmetric or asymmetric presentation. If all replicates are in error, the error is said to be Symmetric, otherwise it is Asymmetric. In addition to timing errors, which have their intuitive meaning, replicates can differ in value and presence.

- **Value Errors.** If one or more of the values of the replicates differs from other replicates, two errors are possible, depending on the test for equality used: if any variation in replicates produces different behaviors (e.g., if the inputs are used in a hash function) then an AsymmetricExactValue error is present. Otherwise the most appropriate error type is AsymmetricApproximateValue.
- **Omission Errors.** Errors of omission can occur in replicated services either symmetrically or asymmetrically. We further distinguish between the omission of individual service items and entire services.

The AADL Error Library: A Case Study

The AADL Error Library is not typically used on its own. Instead, it is most often used as a piece of the larger safety analysis process in safety-critical engineering projects. In addition to efforts to analyze or demonstrate the error annex, a number of larger projects have used the ontology in a range of settings, typically in the avionics domain. These include post-hoc analysis of an industrial safety accident, extensive analysis of an individual component, and large-scale feasibility demonstration in a military shadow project.

The SEI published a detailed description of a typical use of the error ontology in a report on the aforementioned military shadow project. Steps 0, 1, and 3 (in the process below) compose the first phase of the effort: identifying *which* error types to use. Steps 2 and 4 compose the second phase: specifying *how* the error types are consumed and produced. This second phase--essentially specifying the input and output error specifications of each component--is what takes advantage of both the propagation paths (created during instantiation) and the type system that underlies the library to enable forward and backward reasoning.

(0) At a high level, the safety process (in the shadow project) began with a functional and hardware model of a system specified in AADL. The ontology was used, along with the functional model of the system, to drive conversations between a safety analyst and a domain expert.

(1) The output of this conversation was a collection of error types that had been customized to the domain (e.g., the ServiceOmissionerror type was extended into a specific subsystem failure). One of the most interesting error discoveries was that information on the system's location traveled along two paths, one of which had a significantly longer computation time. This led to the discovery of an AsymmetricTimingError.

(2) All of the findings from the dialogue were operationalized by extending error types from the ontology into a system-specific error library. These types were then used to drive a hazard analysis of the system.

(3) Steps 1 and 2 were then repeated for the hardware model.

(4) Finally, interactions between the hardware and functional model were analyzed using the combined set of system-specific error types to determine the final error behavior of the system.

Looking Ahead

Both AADL and its error modeling annex continue to evolve, and there is more work to do in the future, including in the following areas:

- **Safety and Security.** There is growing interest in the overlap of security analysis with traditional safety assessment tasks. The extent to which the Error Library, and the EMV2 itself, can support these tasks is an open area of research. While we are encouraged that most security issues eventually manifest in one of the same error types as traditional safety concerns, others, such as the inadvertent leaking of privileged data, have no safety equivalent and will likely require modifications to the library.
- **Limitations of an Effects Focus.** A strict focus on the effects of errors is one of the strengths of the Error Library, though there may be limits to this approach. To this end, we are experimenting with less effects-focused error families that deal with, for example, concurrency issues. The costs,

benefits, and tradeoffs involved in expanding beyond a strict effects focus is not yet clear, however, so more study needs to be done.

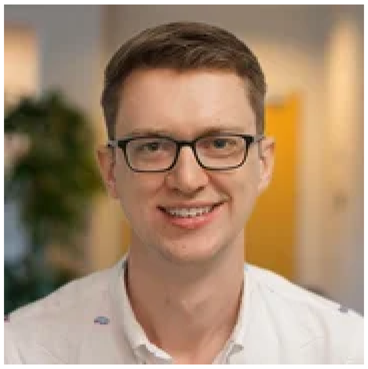
- **Patterns for Co-Occurring Errors.** System behavior in the presence of errors can become arbitrarily complex when those errors can co-occur, so a concise way of specifying co-occurrence is desirable. The Error Library's set of replication errors (i.e., the creation of wholly new types, see Section 5.4 in the paper) is one attempt at this, as is the more general error type product construct, which creates a new error type from two extant types (e.g., a value error and a time error). We are still researching when one specification mechanism is preferable to another.

Additional Resources

Read the paper from which this blog post is adapted, *The AADL Error Library: An Operationalized Taxonomy of System Errors.*

Read the SEI Special Report *Architecture-Led Safety Analysis of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System*.

WRITTEN BY



Sam Procter

DIGITAL LIBRARY PUBLICATIONS

SEND A MESSAGE

A Model-Based Tool to Assist in the Design of Safety-Critical Systems

MARCH 7, 2022 • BY SAM PROCTER

Integrating Safety and Security Engineering for Mission-Critical Systems

MAY 10, 2021 • BY SAM PROCTER, SHOLOM G. COHEN

Simultaneous Analysis of Safety and Security of a Critical System

SEPTEMBER 11, 2017 • BY SAM PROCTER

MORE IN SOFTWARE ARCHITECTURE

Building Quality Software: 4 Engineering-Centric Techniques

AUGUST 19, 2024 • BY ALEJANDRO GOMEZ

The OSATE Slicer: Fast Reachability Query Support for Architectural Models

NOVEMBER 13, 2023 • BY SAM PROCTER

How to Use Docker and NS-3 to Create Realistic Network Simulations

MARCH 27, 2023 • BY ALEJANDRO GOMEZ

Software Isolation: Why It Matters to Software Evolution and Why Everybody Puts It Off

MARCH 20, 2023 • BY MARIO BENITEZ PRECIADO

Experiences Documenting and Remediating Enterprise Technical Debt

DECEMBER 19, 2022 • BY STEPHANY BELLOMO