Safety Analysis with AADL

Julien Delange <jdelange@sei.cmu.edu>
Jerome Hugues <jerome.hugues@isae.fr>

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Objectives

Introduce the AADL Error-Model v2 (EMV2)

Explain main concepts (errors sources and propagation)

Present safety analysis tools

Exercise safety analysis on the ADIRU system
Introduction to the AADL
Error Model Annex v2
Safety Practice in Development Process Context

- **Labor-intensive**
  - Early in system engineering
  - Rarely repeated due to cost

- **Focus on System Engineering Largely**
  - Ignores Software as Hazard Source

**ARP4754A Process**

DO-178B/DO-254 P
ED-12B/ED-80 Process
AADL Error Model Scope and Purpose

System safety process uses many individual methods and analyses, e.g.

- hazard analysis
- failure modes and effects analysis
- fault trees
- Markov processes

Related analyses are also useful for other purposes, e.g.

- maintainability
- availability
- Integrity

Goal: a general facility for modeling fault/error/failure behaviors that can be used for several modeling and analysis activities.

SAE ARP 4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment

Annotated architecture model permits checking for consistency and completeness between these various declarations.
Error Model V2: 4 levels of abstraction

1. Focus on fault interaction with other components
2. Focus on fault behavior of components
3. Focus on fault behavior in terms of subcomponent
4. Types of malfunctions and propagations
Automation of SAE ARP4761 System Safety Assessment Practice

- FHA
  - Spreadsheet
  - Uses error sources

- FMEA
  - Spreadsheet
  - Uses error flows & propagations

- AADL & EMV2
  - Uses error flows & behavior

- FTA
  - CAFTA, OpenFTA
  - Uses composite error behavior

- Markov Chain
  - PRISM
  - Uses error flows & behavior

- RBD/DD
  - OSATE plugin
  - Uses composite error behavior
Value of Automated Architecture-led Safety Analysis

Failure Modes and Effects Analyses are rigorous and comprehensive reliability and safety design evaluations

- Required by industry standards and Government policies
- When performed manually are usually done once due to cost and schedule
- If automated allows for
  - multiple iterations from conceptual to detailed design
  - Tradeoff studies and evaluation of alternatives

Largest analysis of satellite to date consists of 26,000 failure modes

- Includes detailed model of satellite bus
- 20 states perform failure mode
- Longest failure mode sequences have 25 transitions (i.e., 25 effects)
Providing different views

EMV2-like Compositional Fault Behavior Specification for Simulink Models

Figure 9 - Inverse relationship between fault trees (left) and FMEA (right)
Understanding the Cause and Effects of Faults

Through model-based analysis identify architecture induced unhandled, testable, and untestable faults and understand root causes, contributing factors, impact and potential mitigation options.

Fault Impact & FDIR Analysis

- Architecture Fault Model Analysis
- Discover testable and untestable faults
- Discover unhandled faults & safety violations

Faults that can be tested
- Decision coverage

Faults that cannot be tested
- Race conditions

Improved documentation & design

Faults that are unhandled
- Transient data loss
- In protocol

Fault Impact Analysis

- Detection of Unhandled Data Loss Fault
- Root Cause of Data Loss Is Non-deterministic Temporal Buffer Read/Write Ordering

Fault propagation Effects Engine Control Mode to Issue Shut Down Engine Sequence

Read/write Timeline Analysis Under Cyclic Executive & Preemptive Scheduler

Root Cause of Data Loss Is Non-deterministic Temporal Buffer Read/Write Ordering
Safety-Criticality Requirements

Exceptional conditions, anomalies and hazards
- Mode confusion (reported state vs. observed state vs. actual state)
- Unexpected fault conditions and fault impact
- Inclusion/exclusion of pilot in system
- Fault Detection, Isolation, and Recovery (FDIR)
  - Safety system architecture, security system architecture

Certification impact
- Criticality levels, design assurance levels and verification implications
- Partition allocations (isolation) and avoidable certification cost
- Understanding change impact to achieve proportional recertification
Latency Sensitivity in Control Systems

Common latency data from system engineering
- Processing latency
- Sampling latency
- Physical signal latency

Impact of Scheduler Choice on Controller Stability
A. Cervin, Lund U., CCACSD 2006
Software-Based Latency Contributors

- Execution time variation: algorithm, use of cache
- Processor speed
- Resource contention
- Preemption
- Legacy & shared variable communication
- Rate group optimization
- Protocol specific communication delay
- Partitioned architecture
- Migration of functionality
- Fault tolerance strategy
The Symptom: Missed Stepper Motor Steps

Stepper motor (SM) controls a valve

- Commanded to achieve a specified valve position
  - Fixed position range mapped into units of SM steps
- New target positions can arrive at any time
  - SM immediately responds to the new desired position

Safety hazard due to software design

- Execution time variation results in missed steps
- Leads to misaligned stepper motor position and control system states
- Sensor feedback not granular enough to detect individual step misses

Software modeled and verified in SCADE
- Full reliance on SCADE of SM & all functionality
- Problems with missing steps not detected

Software tests did not discover the issue
- Time sensitive systems are hard to test for.

Two Customer Proposed Solutions

Sending of data at 12ms offset from dispatch
- Buffering of command by SM interface
- No analytical evidence that the problem will be addressed
Analysis Results and Solution

Architecture Fault Model Analysis

- Fault impact analysis identifies multiple sources of missed steps
  - Early arrival of step increment commands
  - Step increment command rate mismatch
  - Transient message corruption or loss

- Understanding of error cause
  - When is early too early
  - Guaranteed delivery assumption for step increment commands
Time-sensitive Auto-brake Mode Confusion

Auto-brake mode selection by push button
  • Three buttons for three modes
  • Each button acts as toggle switch

Event sampling in asynchronous system setting
  • Dual channel COM/MON architecture
  • Each COM, MON unit samples separately
    • Button push close to sampling rate results in asymmetric value error
    • COM/MON mode discrepancy votes channel out
    • Repeated button push does not correct problem
    • Operational work around (1 second push) is not fool proof

Avoidable complexity design issue
  • Concept mismatches: desired state by event and sampled event processing

Desirable solution: State communication by multi-position switch
Error Model Annex v2
Main Concepts
Error Type Libraries

Error Type libraries and AADL Packages

- An AADL package can contain one Error Model library declaration
- The **error types** clause represents the Error Type library within the Error Model library
- The Error Type library is identified and referenced by the package name

Error Type library represents a namespace for error types and type sets

- Error type and type set names must be unique within an Error Type library
- An Error Type library can contain multiple error type hierarchies
Error Types & Error Type Sets

*Error type* declarations

```plaintext
TimingError: type;
EarlyValue: type extends TimingError;
LateLate: type extends TimingError;
ValueError: type;
BadValue: type extends ValueError;
```

An *error type set* represents a set of type instances

- Elements in a type set are mutually exclusive
- An error type with subtypes includes instances of any subtype
- A *type product* represents a simultaneously occurring types
  - Combinations of subtypes

```plaintext
InputOutputError : type set {TimingError, ValueError, TimingError*ValueError};
```

An *error type instance*

- Represents the error type of an actual event, propagation, or state

---

*Error Type Set as Constraint*

- \{T1\} tokens of one type hierarchy
- \{T1, T2\} tokens of one of two error type hierarchies
- \{T1*T2\} type product (one error type from each error type hierarchy)
- \{NoError\} represents the empty set

Constraint on state, propagation, flow, transition condition, detection condition, outgoing propagation condition, composite state condition
A Standard Set of Error Propagation Types

Predeclared as library called ErrorLibrary
Includes a common set of aliases
Component Error Propagation

Incoming/Assumed

- Error Propagation
  Propagated errors
- Error Containment:
  Errors not propagated

Outgoing/Contract

- Error Propagation
- Error Containment

Supports Fault Propagation & Transformation Calculus (FPTC) by York University
Also origin of safety cases

Legend

- Port
- Processor
- HW Binding
- Propagated Error Type
- Not propagated

Error Flow through component
Path P1.NoData→P2.NoData
Source P2.BadData
Path processor.NoResource -> P2.NoData

“Not” on propagated indicates that this error type is intended to be contained.
This allows us to determine whether propagation specification is complete.

Bound resources

- Error Propagation
- Error Containment
- Propagation to resource
Error Propagation Declarations

system Subsystem

features
P1: in data port;
P2: in data port;
P3: out data port;

annex EMV2 {**
  use types ErrorLibrary;
  error propagations
  P1: in propagation {NoData, ValueError} ;
P2: in propagation {NoData};
P2: not in propagation  {BadValue};
P3: out propagation {NoData, BadValue};
P3: not out propagation {LateData};
  processor: in propagation {NoResource};
  end propagations; **};

Binding Related Propagation Specifications
Processor, Memory, Connection, Binding, Bindings
Path follows predeclared Binding properties
Error Flows

Error flow specifies the role of a component in error propagation

- The component may be a source or sink of a propagated error types
- The component may pass incoming types through as outgoing types
- The component may transform an incoming type into a different outgoing type
- By default all incoming errors of any feature flow to all outgoing features

The same propagation may be part of a flow source/sink and flow path.

A propagation may be a sink for one type and not for another

```plaintext
annex EMV2 {**
  error propagations
  ... ...
  flows

  es1: error source P3{BadData};
  es2: error source P3{NoData};
  es3: error sink P2{NoData};
  ep1: error path P2{BadData}->P3;  -- same type as incoming type
  ep2: error path P1{ValueError} -> P3{ItemOmission};  -- all value errors xformed into ItemOmission
  ep3: error path processor -> P3

  mapping MyErrorModelLibrary::MyMapping;  -- use a type mapping table

end propagations ; **};
```
Functional Hazard Assessment

Hazard property

- Tailoring for safety standards (ARP4761, MIL-STD-882)
- Associated with error state, error source, outgoing propagation, error type

```plaintext
Hazard property

- Tailoring for safety standards (ARP4761, MIL-STD-882)
- Associated with error state, error source, outgoing propagation, error type

```

```plaintext
device PositonSensor
  features
  PositionReading: out data port DataDictionary::Position;
  flows
  f1: flow source PositionReading {
    latency => 2 ms..3 ms;
  }
  annex EMV2 (**
    use types ErrorLibrary, FMAErrorLibrary;
    use behavior ErrorModLibrary::Simple;
    error propagations
      PositionReading: out propagation {ServiceEmission};
      flows
        cf1: error source PositionReading {ServiceEmission} When Failed;
      end propagations;
  }
  properties
    EMV2::hazards =>
    ( [crossreference] => "1.1.3",
      failure => "Loss of sensor readings",
      phases => "all",
      severity => MILSTD882::Critical;
      likelihood => MILSTD882::Remote;
      description => "No stabilator position readings due to sensor failure"
      comment => "Becomes major hazard, if no redundant sensor";
    )
    applies to cf1.Failed;
  end PositionSensor;
```

```plaintext
Hazard property

- Tailoring for safety standards (ARP4761, MIL-STD-882)
- Associated with error state, error source, outgoing propagation, error type

```

```plaintext
device PositionSensor
  features
  PositionReading: out data port DataDictionary::Position;
  flows
  f1: flow source PositionReading {
    latency => 2 ms..3 ms;
  }
  annex EMV2 (**
    use types ErrorLibrary, FMAErrorLibrary;
    use behavior ErrorModLibrary::Simple;
    error propagations
      PositionReading: out propagation {ServiceEmission};
      flows
        cf1: error source PositionReading {ServiceEmission} When Failed;
      end propagations;
  }
  properties
    EMV2::hazards =>
    ( [crossreference] => "1.1.3",
      failure => "Loss of sensor readings",
      phases => "all",
      severity => MILSTD882::Critical;
      likelihood => MILSTD882::Remote;
      description => "No stabilator position readings due to sensor failure"
      comment => "Becomes major hazard, if no redundant sensor";
    )
    applies to cf1.Failed;
  end PositionSensor;
```
Other Predeclared EMV2 Properties

- Occurrence distribution
  - Distribution functions: Fixed, Poisson/Exponential, Normal/Gauss, Weibull, Binominal
- Persistence: Permanent, Transient, Singleton
- Duration distribution
- Fault kind: design, operational
- State kind: working, nonworking
- Detection mechanism
Consistency in Error Propagation

Mismatched fault propagation and containment assumptions
Discovery of unhandled error propagations.
Software Induced Flight Safety Issue

Original Preliminary System Safety Analysis (PSSA)
System engineering activity with focus on failing components.
Unhandled Hazard Discovery through Virtual Integration

Corrupted data shows airspeed of 2000 knots

Vibration causes data corruption through touching boards

Virtual integration of architecture fault models recording SIL test observations detects unhandled fault.

Response to corrupted airspeed causes stall

Actuator Cmd

NoService Stall

Flight Mgmt System

Auto Pilot
Operational Failed

FMS Processor
Operational Failed

FMS Power

CorruptedData

Airspeed Data

Operat'l Failed

Corrupted

Operat'l Failed

NoData

Operat'l Failed

NoData
Component Error Behavior

Components have error, mitigation, and recovery behavior specified by an error behavior state machine. 

*Transitions* between *states* triggered by *error events* and *incoming propagations*.

Conditions for *outgoing propagations* are specified in terms of the *current state* and *incoming propagations*.

*Detection* of error states and incoming propagations is mapped into a message (event data) with error code in the system architecture model.
Reusable Error Behavior State Machine

annex EMV2 {**

error behavior ExampleBehavior

events
  Fault: error event;
  SelfRepair: recover event;
  Fix: repair event;

states
  Operational: initial state ;
  FailStopped: state;
  FailTransient: state;

transitions
  SelfFail: Operational -[Fault]-> (FailStopped with 0.7, FailTransient with 0.3);
  Recover: FailTransient -[SelfRepair]-> Operational;

end behavior;

Properties
  EMV2::OccurrenceDistribution => [ ProbabilityValue => 0.00004 ; Distribution => Poisson;]

  applies to Fault;
Component Error Behavior Specification

Component-specific behavior specification

- Identifies an error behavior state machine
- Optionally defines component specific error events
- Specifies transition trigger conditions in terms of incoming propagated errors or working condition of connected component
- Specifies propagation conditions for outgoing propagated errors in terms of states & incoming propagated errors
- Specifies detection conditions under which becomes an event with error code in the core AADL model

```plaintext
use types ErrorLibrary;
use behavior MyErrorLibrary::ExampleBehavior;
component error behavior
transitions  -- additional transitions that are component specific
  Operational-[Port1{NoData} and Port2{NoError}]->FailTransient;
  FailStopped-[port1{BadData}];
propagations
  all -[2 ormore (Port1{BadData}, Port2{BadData},Port3{BadData})]-> Outport3(BadData);
detections
  FailedState -[]-> Self.Failed (FailCode) ;  -- Could also report on an outgoing error port
properties
  EMV2::OccurrenceDistribution => [ ProbabilityValue => 0.00005 ; Distribution => Poisson;]
  applies to Fault; -- component specific occurrence value
end behavior;
```
Error Model at Each Architecture Level

- Abstracted error behavior of FMS
  - Error behavior and propagation specification

  \[ \text{[1 ormore(} \text{FG1.Failed or AP1.Failed)} \text{ and } \text{1 ormore(} \text{FG2.Failed or AP2.Failed)} \text{ or AC.Failed] -> Failed} \]

- Composite error behavior specification of FMS
  - State in terms of subcomponent states

Consistency Checking Across Levels of the Hierarchy
Error Model Annex v2
Safety Analysis tools
AADL & Safety Evaluation – Tool Overview

**FHA**
- Spreadsheet
- Use error propagations

**FTA**
- CAFTA
- OpenFTA
- Use composite behavior
- Error flows

**Markov Chain**
- PRISM
- Use error flow
- Error behavior

**FMEA**
- Spreadsheet
- Error behavior
- Propagations
Safety Analysis & AADL

Preliminary System Safety Assessment (PSSA) support
- High-level component, interfaces from the OEM
- Automatic generation of validation materials (FHA, FTA)

System Safety Assessment (SSA) support
- Use refined models from suppliers
- Enhancement of error specifications
- Support of quantitative safety analysis (FTA, FMEA, MA)
Evolution of Safety Analysis process with AADL

Preliminary System Safety Assessment

Component types (system interfaces) AADL

Component implementation AADL

Validation Materials (FHA, FTA)

Check PSSA and SSA consistencies

Validation with quantitative fault rates (FMEA, FTA, DD, MA)

System Safety Assessment

Refinement & development evolution
Safety Analyses on Refined Architecture

Aircraft-Level Safety Analysis
- Define aircraft failure conditions
- Allocate failure to system functions
- Perform PSSA and SSA

Avionics Subsystem Level Safety Analysis
- Perform PSSA and SSA at subsystem level
- Ensure consistency with aircraft level analysis

Navigation Sub-Subsystem Level Safety Analysis
- Perform PSSA and SSA at sub-subsystem level
- Ensure consistency with aircraft level analysis
Evolution of the AADL model

Component extension, refinement & implementation

AADL model Version n

AADL model Version n + 1

Development Process
Evolution of Safety Assessment with AADL

AADL model version n

Automatic Fault-Tree Generation

FTA version n

FTA refinement & improvement

AADL model version n + 1

Automatic Fault-Tree Generation

FTA version n + 1

Development Process
Functional Hazard Analysis Support

Use of **component error behavior**
- Error propagations rules
- Internal error events

Specify initial failure mode

Define error description and related information

Create spreadsheet containing FHA elements
- To be reused by commercial or open-source tools
Fault-Tree Analysis Support

Use of composite error behavior
FTA nodes

Use of component error behavior
Incoming error events

Walk through the components hierarchy
Generate the complete fault-tree
Focus on specific AADL subcomponents

Export to several tools
Commercial: CAFTA
Open-Source: EMFTA, OpenFTA
Failure Mode and Effects Support

Use of component error behavior
   - Error propagations rules (source, sink, etc.)
   - Internal error events

Traverse all error paths
   - Record impact over the components hierarchy

Use error description and related information

Create spreadsheet containing FHA elements
   - To be reused by commercial or open-source tools
Reliability Block Diagram
aka ARP4761 Dependence Diagram (DD)

Use of **composite error behavior**
- Error propagations rules (source, sink, etc.)
- Internal error events

Compute reliability of the Dependence Diagram
- Use of recover and failure events
- Overall probability of system failure

Support in OSATE (built-in)
Error Model Annex v2
Application to the ADIRU
Annotating the model with Error Information (1)

Declaring error sources

Documenting the error

```plaintext
device implementation acc_device.impl

use types ADIRU_errLibrary;
use behavior ADIRU_errLibrary::simple;

error propagations
accData : out propagation{ValueErroneous};
flows
f1 : error source accData{ValueErroneous} when failed;
end propagations;

properties
emv2::hazards =>
{{
crossreference => 'N/A';
failure => "Accelerometer value error";
phases => ("in flight");
description => "Accelerometer starts to send an erroneous value";
comment => "Can be critical if not detected by the health monitoring";
}}
applies to accData.valueErroneous;

EMV2::OccurrenceDistribution => { ProbabilityValue => 3.4e-5 ; Distribution => Fixed;}
applies to accData.valueErroneous;
end acc_device.impl;
```
Annotating the model with Error Information (2)

Passing the error directly through components features
Receiving a erroneous value makes the component to fail
### Functional Hazard Assessment

List all potential error sources

Include documentation from the model

Required by ARP4761 safety standard

<table>
<thead>
<tr>
<th>Component</th>
<th>Error</th>
<th>Hazard Description</th>
<th>ossefer</th>
<th>Functional Failure</th>
<th>Operational Phases</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc1</td>
<td>&quot;ValueErrorous on accData&quot;</td>
<td>&quot;Accelerometer starts to send an erroneous value&quot;</td>
<td>&quot;N/A&quot;</td>
<td>&quot;Accelerometer value error&quot;</td>
<td>&quot;in flight&quot;</td>
<td>&quot;Can be critical if not detected by the health monitoring&quot;</td>
</tr>
<tr>
<td>acc2</td>
<td>&quot;ValueErrorous on accData&quot;</td>
<td>&quot;Accelerometer starts to send an erroneous value&quot;</td>
<td>&quot;N/A&quot;</td>
<td>&quot;Accelerometer value error&quot;</td>
<td>&quot;in flight&quot;</td>
<td>&quot;Can be critical if not detected by the health monitoring&quot;</td>
</tr>
<tr>
<td>acc3</td>
<td>&quot;ValueErrorous on accData&quot;</td>
<td>&quot;Accelerometer starts to send an erroneous value&quot;</td>
<td>&quot;N/A&quot;</td>
<td>&quot;Accelerometer value error&quot;</td>
<td>&quot;in flight&quot;</td>
<td>&quot;Can be critical if not detected by the health monitoring&quot;</td>
</tr>
<tr>
<td>acc4</td>
<td>&quot;ValueErrorous on accData&quot;</td>
<td>&quot;Accelerometer starts to send an erroneous value&quot;</td>
<td>&quot;N/A&quot;</td>
<td>&quot;Accelerometer value error&quot;</td>
<td>&quot;in flight&quot;</td>
<td>&quot;Can be critical if not detected by the health monitoring&quot;</td>
</tr>
<tr>
<td>acc5</td>
<td>&quot;ValueErrorous on accData&quot;</td>
<td>&quot;Accelerometer starts to send an erroneous value&quot;</td>
<td>&quot;N/A&quot;</td>
<td>&quot;Accelerometer value error&quot;</td>
<td>&quot;in flight&quot;</td>
<td>&quot;Can be critical if not detected by the health monitoring&quot;</td>
</tr>
<tr>
<td>acc6</td>
<td>&quot;ValueErrorous on accData&quot;</td>
<td>&quot;Accelerometer starts to send an erroneous value&quot;</td>
<td>&quot;N/A&quot;</td>
<td>&quot;Accelerometer value error&quot;</td>
<td>&quot;in flight&quot;</td>
<td>&quot;Can be critical if not detected by the health monitoring&quot;</td>
</tr>
</tbody>
</table>
Fault Impact Analysis

Bottom-up approach

Trace the error flow defined in the architecture

Required by ARP4761 safety standard
Error Model Annex v2
Conclusion
Architecture Fault Modeling Summary

Architecture Fault Modeling with AADL

- Error Model Annex was originally published in 2006
  - Supported in AADL V1 and AADL V2
- Standardized Error Model Annex (V2) based on user experiences
- Error Model V2 concepts and ontology can be applied to other modeling notations

Safety Analysis and Verification

- Error Model Annex front-end available in OSATE open source toolset
  - Allows for integration with in-house safety analysis tools
- Multiple tool chains support various forms of safety analysis (Honeywell, Aerospace Corp., AVSI SAVI, ESA COMPASS, WW Technology)
- FHA, FMEA, fault tree, Markov models, stochastic Petri net generation from AADL/Error Model
References

Website  [www.aadl.info](http://www.aadl.info)

Public Wiki  [https://wiki.sei.cmu.edu/aadl](https://wiki.sei.cmu.edu/aadl)

EMFTA  [https://github.com/juli1/emfta](https://github.com/juli1/emfta)


