An Overview of AADL and Toolsets to Support the Engineering of Safety-critical Systems

John Hudak, Jerome Hugues
SSD/ACPS/MBE
An Overview of AADL and Toolsets to Support the Engineering of Safety-critical Systems

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

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

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

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

DM21-0067
Outline

Model Based Engineering at the SEI
Model-Based for Software-intensive systems using AADL
• Why AADL?
• MBSE .. a collection of standards
• AADL Overview
• Wrap-up
Outline

Model Based Engineering at the SEI
Model-Based for Software-intensive systems using AADL
Making Critical Systems Safer and More Secure

Modern embedded systems need to be both safe and secure. As we have seen, the pace and scale of the development of these systems means traditional methods cannot keep up.

Research to Practice
The SEI works to rapidly move ideas from research in embedded systems – conducted either here at the SEI, in academia, or in industry – to practice.
Model-Based Engineering for Cyber-Physical Systems

Create the best design that holds up over time as the system evolves. + Test the design without having to write any code. = Build a single model to assess hardware and embedded software before the system is built.

**SAE AADL / ACVIP**
Standardized language and process for the engineering safety-critical systems.

**OSATE**
Open Source AADL toolset for performing verification and validation (V&V).

**DoD Transitioning**
Maturity increased through pilot projects and trainings.
Before You Even Write a Line of Code…

AADL allows you to design the entire system and see where the problems may occur. Then you can change the design of the system to eliminate those errors.

Being able to perform a virtual integration of the software, hardware, and system is the key to identifying problems early – and changing the design to ensure those problems will not occur.

About AADL

- SAE Avionics AADL standard adopted in 2004
- Focused on embedded software system modeling, analysis, and generation
- Strongly typed language with well-defined semantics
- Used for critical systems in domains such as avionics, aerospace, medical, nuclear, automotive, and robotics
Outline

Model Based Engineering at the SEI

Model-Based Engineering for Software-intensive systems using AADL

• Challenges of embedded systems – AADL to the rescue Why AADL?
• Model-Based System Engineering, AADL
• AADL Language Overview
• AADL Tooling
We Rely on Software for Safe System Operation

Quantas Airbus A330-300 Forced to make Emergency Landing - 36 Injured

By Ed Johnson
Oct 15 (Bloomberg) -- A Qantas Airbus A330-300, after Australian investigators found the flight simulator had failed, switched off its jet to nowhere.

The Airbus A330-300 was on a flight from Singapore to Perth when it suddenly changed altitude during a flight from Singapore to Perth, Qantas said.

Thirty-six passengers and crew were injured, some seriously, in a mid-air drama that forced a Qantas jetliner to make an emergency landing, the Australian carrier and police said on Tuesday.

The terrifying incident saw the Airbus A330-300 issue a mayday call when it suddenly changed altitude during a flight, before the pilots regained control.

This appears to be a unique event," the bureau said, adding that offshore, France-based Airbus, the world's largest maker of commercial aircraft, issued a tell-all note to airlines that fly A330s and A340s fitted with the same air-data computer. The advisory is "aimed at minimizing the risk in the unlikely event of a similar occurrence."

FAA says software problem with Boeing 787s could be catastrophic

By Dan Catchpole
@dcatchpole

The Federal Aviation Administration says a software problem with Boeing 787 Dreamliners could lead to one of the advanced jetliners losing electrical power in flight, which could lead to loss of control.

The FAA notified operators of the airplane Friday that if a 787 is powered continuously for 248 days, the plane will automatically shut down its alternating current (AC) electrical power.

Embedded software systems introduce a new class of problems not addressed by traditional system safety analysis

Breakdown in human intensive safety assessment process
The Safety-Critical Embedded Software System Challenge

Problem:
Software increasingly dominates safety and mission critical system development cost. 80% of issues discovered post unit test.

Solution: Early discovery of system level issues through architecture modeling, virtual Integration and incremental analytical assurance.

Approach:
International standard based research driven technology matured into practice through pilot projects and industry initiatives. (SAE International Aerospace Standard AS-5506B)
Development of an open source research prototyping platform continually enhances analysis, verification, and generation capabilities.

Reducing Defect Leakage through Early Analytical Assurance is Critical
High Fault Leakage Drives Major Increase in Rework Cost

Aircraft industry has reached limits of affordability due to exponential growth in SW size and complexity.

70% Requirements & system interaction errors introduced 3.5% are detected, 1x cost to remove

10% errors introduced 80% late error discovery, 20-100x cost to fix

Major cost savings through rework avoidance by early discovery and correction
A $10k architecture phase correction saves $3M

Where faults are introduced
Where faults are found

The estimated nominal cost for fault removal

Sources:

80% late error discovery, 20-100x cost to fix
20.5% 300-1000x
0%, 9% 80x
20%, 16% 5x
20%, 50.5% 20x
80% late error discovery, 20-100x cost to fix

20.5% 300-1000x
0%, 9% 80x
20%, 16% 5x
20%, 50.5% 20x
80% late error discovery, 20-100x cost to fix

Total System Cost
Boeing 777 $12B
F-35 $59B

Software as % of total system cost
1997: 45% → 2010: 66% → 2024: 88%

Post-unit test software rework cost
50% of total system cost and growing

Where faults are introduced
Where faults are found

The estimated nominal cost for fault removal

Sources:
Technical Challenges in Safety-Critical Embedded Software Systems

Why do system level failures still occur despite best safety practices?

Need Semantics To Address these Challenges

Measurement units Boolean, integer, real, vs data abstraction

Embedded software systems have become a major safety and cyber security risk

An Overview of AADL and Toolsets to Support the Engineering of Safety-critical Systems
© 2021 Carnegie Mellon University

DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.
Analysis of System Properties via Architecture Model
A Contribution to Single Source of Truth

Change of Encryption from 128 bit to 256 bit

Resource Consumption
- Bandwidth
- CPU Time
- Power Consumption

Higher CPU Demand

Increased Latency

Real-Time Performance
- Deadlock/
- Starvation
- Latency
- Execution Time/
- Deadline

Affects Temporal Correctness

Data Quality
- Temporal Correctness
- Data Precision/
- Accuracy
- Confidence

Cyber Security
- Availability
- Integrity
- Confidentiality

Safety & Reliability
- Hazard Analysis
- FMEA
- FTA
- MTRF

Potential New Hazard

One change drives multiple system issues

SAE AS5506 AADL

Single Source of Truth Across Analysis Models
Outline

Model Based Engineering at the SEI

Model-Based Engineering for Software-intensive systems using AADL

- Challenges of embedded systems – AADL to the rescue Why AADL?
- Model-Based System Engineering, AADL
- AADL Language Overview
- AADL Tooling
Model-Based System/Software Engineering

Overarching objectives

MBSE complements typical software programming with *models* to

1. Organize stakeholders needs and elicit requirements

2. Capture system elements – design, reverse engineering or COTS
   - Interfaces, components internals (static and behavioral), and
   - A system architecture built from those: deployment, (re-)configuration

3. Apply analytical frameworks to assess model’s “compliance to some objectives”
   - Syntactic, conformance to guidelines, patterns
   - Quality of system, w.r.t. performance, safety, security, behavior metrics

4. Synthesize portions of software from models
   - E.g. functional: Simulink, SCADE; Architectural: UML, AADL
   - No synthesis or link to code in SysML, as SysML has only high-level concepts

Models as processable artifacts to guide the software engineering process

"Modeling is the new programming"

Provide more insights than code-only solution through relevant abstractions and automation
MBSE – Not just SysML

Model-based systems engineering (MBSE) is the “*formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases.*” (INCOSE 2007)

SysML support capturing relationships among system functions, requirements, developers, and users. But not the later development stages

Other modeling notations are required to

Capture hardware platforms, software architecture, behavioral semantics, deployment of SW to HW, support safety or security assessment, performance analysis, behavioral verification, memory budget validation, etc...
Architecture Analysis & Design Language (AADL) Standard Targets Embedded Software Systems

AADL captures mission and safety critical embedded software system architectures in virtually integrated analyzable models to discover system level problems early and construct implementations from verified models.

In 2008 Aerospace industry initiative chose AADL over SysML and other notations as it specifically addresses embedded software systems.

SAE International AS 5506 Standard Suite
Standards provide long-term industry-wide solutions to support multi-organization model-based engineering.
Not just SysML vs AADL, larger aggregation of standards

Filling the Modeling and Analysis Gap for Embedded Software System
Outline

Model Based Engineering at the SEI

Model-Based Engineering for Software-intensive systems using AADL
- Challenges of embedded systems – AADL to the rescue Why AADL?
- Model-Based System Engineering, AADL
- AADL Language Overview
- AADL Tooling
An Overview of AADL and Toolsets to Support the Engineering of Safety-critical Systems

© 2021 Carnegie Mellon University

Distribution Statement A: This material has been approved for public release and unlimited distribution.

Core AADL language standard [V1 2004, V2 2012, V2.2 2017]

- Focused on *embedded software system modeling, analysis, and generation*
- Strongly typed language with well-defined semantics for execution of threads, processes on partitions and processor, sampled/queued communication, modes, end to end flows
- Textual and graphical notation, XML/XMI interface to ease processing by 3rd party tool
- V3 in progress: interface composition, system configuration, binding, type system unification
- [http://aadl.info](http://aadl.info)

Ongoing work to align AADL and SysML in a common workflow -> SEI, Adventium Labs, ANSYS

<table>
<thead>
<tr>
<th>Standardized AADL Annex Extensions</th>
<th>AADL Annexes in Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Modeling extension for interfacing with data models (UML, ASN.1, …) [2011]</td>
<td>Requirements Definition and Assurance Annex</td>
</tr>
<tr>
<td>FACE Annex [2019]</td>
<td></td>
</tr>
</tbody>
</table>
What are AADL Components?

Application Components

**System** – hierarchical organization of components

**Process** – protected address space

**Thread** – a schedulable unit of concurrent execution

**Thread group** – logical organization of threads

**Data** – potentially sharable data

**Subprogram** – callable unit of sequential code

Execution Platform & Device Components

**Processor / Virtual Processor** – Provides thread scheduling and execution services

**Memory** – provides storage for data and source code

**Bus / Virtual Bus** – provides physical/logical connectivity between execution platform components

**Device** – interface to external environment
What does AADL actually look like?

Semi-formal semantics

Only architectural elements

Annexes add functionality:
- Error Modeling
- Behavior
- Code Generation

```plaintext
79  |  Basic/naive version that abstracts all the valves with
80  |  a selector subsystem. This selector subsystem hides
81  |  the physical logic behind the selector, shutoff and
82  |  meter/anti-skid valves.
83  |  
84  |  system implementation wbs.basid extends wbs.generic
85  |  
86  |  subcomponents
87  |  bscu : refined to system impl::bscu::bscu.basic;
88  |  -- The selector subsystem
89  |  selector : refined to system impl::valves::selector_basic{Classifier_Substitution_Rule => Type_Ext
90  |  wheel : refined to system impl::wheel::wheel_one_input.i{Classifier_Substitution_Rule => Type_E}
91  |  
92  |  connections
93  |  blue_to_selector : bus access blue.pump.pressure_output <-> selector.blue_input;
94  |  green_to_selector : bus access green.pump.pressure_output <-> selector.green_input;
95  |  
96  |  bscuSel_to_selector : port bscu.Select.Alternate -> selector.select_alternate;
97  |  bscu_cmdnor_to_selector : port bscu.cmd.nor -> selector.cmd.nor;
98  |  bscu_cmdalt_to_selector : port bscu.cmd.alt -> selector.cmd.alt;
99  |  
100 |  selector_to_wheel : port selector.output <-> wheel.input;
101 |  end wbs.basid;
```
Textual and Graphical Representation Example

```aadl
system CarSystem
end CarSystem;
data MyBrakeData
end MyBrakeData;
device BrakePedal
  features
    samples: out data port MyBrakeData;
end BrakePedal;

system BrakingSystem
  features
    brake_data: in data port MyBrakeData;
end BrakingSystem;

system implementation CarSystem.impl
  subcomponents
    braking: system BrakingSystem;
    brake_pedal: device BrakePedal;
  connections
    c1: port brake_pedal.samples -> braking.brake_data;
end CarSystem.impl;
```

What system-level requirements do we want to verify?
- Signal/data latency
- Data interface consistency
- Data flow/dependency
- Security-Data Confidentiality
- Tread scheduling
- Thread binding/CPU loading
- Memory usage
- Hazards & error flow
This AADL model represents threads executing within processes (dedicated address spaces), the communications and connections among the components, and the binding of threads and processes to computer resources (e.g., threads to the processor on which they execute).
AADL: Modeling in the small

AADL modeling components: precise execution semantics

- **Software:** thread (group), process, data, subprogram (group),
- **Hardware:** processor, memory, bus, device, virtual processor, virtual bus
- **Composite:** system, abstract

Supports system concepts of continuous control & event response processing

- Data and event flows, call/return, shared access
- End-to-End flow specifications

(from AADLv2 standard)
AADL: Modeling in the large

Operational modes & fault tolerant configurations

- **Modes** & mode transition

Modeling of large-scale systems

- Component variants (**refine**) layered system modeling, **packages**, **abstract**, **prototype**, parameterized templates, **arrays** of components, **connection** patterns

Accommodation of diverse analysis needs

- Extension mechanism (**extends**), standardized extensions
Outline

Model Based Engineering at the SEI

Model-Based Engineering for Software-intensive systems using AADL

• Challenges of embedded systems – AADL to the rescue Why AADL?
• Model-Based System Engineering, AADL
• AADL Language Overview
• AADL Tooling
AADL capabilities

AADL is highly tunable, with a restricted set of concepts
• Demonstrated many use cases, 1600+ academic publications

AADL as a backbone, federating multiple activities
• Analysis through generation of intermediate models + external tools

Non exhaustive list of analysis capabilities
• Integration to a process: with SysML, Simulink, SCADE
• Architectural pattern checks:
  MILS, ARINC, Ravenscar, Synchronous
• Model checking:
  • Timed/Stochastic/Colored Petri Nets
  • Timed automata et al.: UPPAAL, Versa, TASM
• Scheduling: MAST, Cheddar, CARTS
• Performance evaluation: real-time and network calculus
• Fault analysis: COMPASS, Stochastic Petri Nets, PRISM
• Simulation: ADeS, Marzhin
• Energy consumption of SoC: OpenPeople project
• Code generation: SystemC, C, Ada, RTSJ, Lustre
• WCET analysis: mapping to Bound-T

AADL demonstrated its suitability to support various analysis for the real world
AADL commercial toolchains

Multiple AADL toolchains exist, they can be easily combined thanks to the textual syntax. Examples:

- **OSATE (SEI/CMU)** [https://osate.org/](https://osate.org/)
  - Eclipse-based tools. Reference implementation
  - Textual and graphical editors + various plug-ins for latency, processor utilization, memory utilization, data consistency, security, safety analysis (MIL STD 882E, ARP4761), ARINC653

- **CAMET** (Adventium Lab) [https://www.adventiumlabs.com/curated-access-model-based-engineering-tools-camet-library](https://www.adventiumlabs.com/curated-access-model-based-engineering-tools-camet-library)
  - Extensions to OSATE to support other analysis (Multiple Independent Levels of Security (MILS), Framework for Analysis of Schedulability, Timing and Resources (FASTAR))

  - Eclipse-based tools. Combine SysML, AADL and other formalisms, code generation

- **AADL Inspector** (Ellidiss) [https://www.ellidiss.com/products/aadl-inspector/](https://www.ellidiss.com/products/aadl-inspector/)
  - Lightweight editor, model simulation, scheduling analysis
Wrap-Up

Model-based systems engineering (MBSE) is the “formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases.” (INCOSE 2007)

SysML support capturing relationships among system functions, requirements, developers, and users. But not the later development stages of Systems Engineering Prediction of runtime characteristics at different fidelity

Analyzeable models to drive development

AADL captures at a high-level: hardware platforms, software architecture, behavioral semantics, deployment of SW to HW

AADL supports safety or security assessment, performance analysis, behavioral verification, memory budget validation, etc...

As a Standard, common modeling notation across organizations
Outline

Introduction to SEI

Model-Based for Software-intensive systems using AADL

AADL in practice
Ideal Systems Engineering Process from models to code

Let’s assume we plan at implementing our own UAV control logic, using the Crazyflie as an example

Steps
• One high-level requirement: piloting the UAV
• Modeling the functional chain
• Refining it down to a logical and physical chain

And then performing verification, validation and simulation

Using AADL as backbone for all models
AADL in practice: Bitcraze Crazyflie

Lightweight UAV by Bitcraze

• [https://www.bitcraze.io/crazyflie-2/](https://www.bitcraze.io/crazyflie-2/)

Hardware:

• 2 x MCU : Cortex-M4 + nRF51822
• IMU: MPU-9250 + Pressure Sensor LPS25H
• Bluetooth Low Energy radiocommunication

Software:

• Regular control/command loop
• Manually implemented, in C + FreeRTOS
Crazyflie architectures

Ideal Systems Engineering Process from models to code

Define a model-based process that
Ideal Systems Engineering Process from models to code

Define a model-based process that

- Separates concerns
  - Runtime architecture ::= configured middleware
  - Functional architecture ::= components
Define a model-based process that

- Separates concerns
  - Runtime architecture ::= configured middleware
  - Functional architecture ::= components
- Mitigates impact of functional code
  - Using safe functional design approach
  - Visible interface and metrics, bounded risks
Ideal Systems Engineering Process from models to code

Define a model-based process that

• Separates concerns
  - Runtime architecture ::= configured middleware
  - Functional architecture ::= components
• Mitigates impact of functional code
  - Using safe functional design approach
  - Visible interface and metrics, bounded risks
• Leverages architectural models to
  - Generate middleware from architecture
Ideal Systems Engineering Process from models to code

Define a model-based process that
• Separates concerns
  - Runtime architecture ::= configured middleware
  - Functional architecture ::= components
• Mitigates impact of functional code
  - Using safe functional design approach
  - Visible interface and metrics, bounded risks
• Leverages architectural models to
  - Generate middleware from architecture
  - Enable formal methods
    • Both at model-level and implementation-level

Requirements, Design, ..
Middleware
Detailed design
Application Code
Run-time Arch.
Functional Arch.
Ideal Systems Engineering Process from models to code

Define a model-based process that
• Separates concerns
  - Runtime architecture ::= configured middleware
  - Functional architecture ::= components
• Mitigates impact of functional code
  - Using safe functional design approach
  - Visible interface and metrics, bounded risks
• Leverages architectural models to
  - Generate middleware from architecture
  - Enable formal methods
    • Both at model-level and implementation-level
• Connects to Model-Based Systems Engineering

Certification as long term objective
Using AADL – Outline

1. Modeling Architectures
AADL Functional chain library

Build a package with all functions as abstract components

![Diagram of AADL Functional chain library](image-url)
AADL functional chain implementation

Abstract data

Function as abstract component
Crazyflie Hardware

Same strategy: library of components + system implementation
Crazyflie Complete System

Software Processes

Hardware/Software Binding
Functional / Physical bindings (allocation)
Using AADL – Outline

1. Modeling Architectures: AADL
2. Architectural pattern enforcement
Architectural pattern enforcement

AADL is an Architecture Design Language
• What about an Analysis Design Language?

Use cases: define project-specific analysis
• Enforcing architectural constraints
  - E.g. Ravenscar, security, ARINC653, OS configuration, etc.
• Evaluating contracts, e.g. compatibility of configuration
  - E.g. controller implementation must be triggered at the right period

Use of Resolute, by Collins, inherit from REAL: Requirement Enforcement and Analysis Language by Olivier Gilles PhD thesis
• A DSL to check static invariants: patterns, contracts, requirements
• Project-specific analysis, mining architecture models
Architectural constraints – Example

Ravenscar profile for mono-core systems correctly applied

```
system ravenscar_sys
annex resolute {**
prove ravenscar_rule_component(this) -- Implementation must match Ravenscar constraints **};
end ravenscar_sys;

is_Scheduling_Configured(c: component) <=
** "Thread " c " is correctly configured" **
has_property(c, Timing_Properties::Compute_Execution_Time) and -- Capacity
has_property(c, Timing_Properties::Period) and -- Period
has_property(c, Timing_Properties::Deadline) and -- Deadline
has_property(c, Thread_Properties::Priority) -- Priority
```
Using AADL – Outline

1. Modeling Architectures: AADL
2. Architectural pattern enforcement: Resolute
3. Code generation and middleware
Code generation and middleware
Architecture-centric process

AADL captures tasks, queues, buffers, protocols
⇒ **Generate** middleware stack on top of minimal runtime/real-time OS
Code generation and middleware: Ocarina

http://www.openaadl.org

Contributions are supported by tools

1. Ocarina: AADL model “compiler”, FLOSS
   • Compiler architecture, AADL front-ends, code generation backends

2. PolyORB-HI runtimes
   • Ada High-Integrity profiles, with Ada native and bare board runtimes
   • C POSIX or RTEMS, for RTOS & Embedded,
   • Time and Space partitioning, e.g. ARINC653 C APEX, AIR, Xtratum

Generated code quality tested in various contexts

• WCET, quality, code coverage, etc.
• Meet High-Integrity coding profiles
  - Ravenscar model of computations, static configuration of all elements (memory, buffers, tasks, drivers, etc.), no dynamicity
Using AADL – Outline

1. Modeling Architectures: **AADL**
2. Architectural pattern enforcement: **Resolute**
3. Code generation and middleware: **Ocarina and PolyORB-HI**
4. V&V strategy
   a) Code level
   b) Model-level
V&V Strategy – Code-level

Use SPARK2014 Ada extensions to annotate runtime

- High-Integrity profile, Ravenscar Model of Computation
- Type invariants, data flows, pre/post conditions

<table>
<thead>
<tr>
<th>SPARK Analysis results</th>
<th>Total</th>
<th>Flow</th>
<th>Provers</th>
<th>Justified</th>
<th>Unproved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Dependencies</td>
<td>35</td>
<td>32</td>
<td>.</td>
<td>3</td>
<td>.</td>
</tr>
<tr>
<td>Flow Dependencies</td>
<td>6</td>
<td>6</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>Initialization</td>
<td>58</td>
<td>55</td>
<td>.</td>
<td>3</td>
<td>.</td>
</tr>
<tr>
<td>Run-time Checks</td>
<td>281</td>
<td>.</td>
<td>281 (CVC4 93%, Trivial 7%)</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>Assertions</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>Functional Contracts</td>
<td>102</td>
<td>.</td>
<td>63 (CVC4 75%, Trivial 25%)</td>
<td>39</td>
<td>.</td>
</tr>
<tr>
<td>Total</td>
<td>496</td>
<td>93 (19%)</td>
<td>344 (69%)</td>
<td>59 (14%)</td>
<td>.</td>
</tr>
</tbody>
</table>
V&V Strategy – Model-level

Requirement to assess model-level properties
- Prior to code generation, ensures model is “safe”
- Exploitation of AADL static and behavioral semantics

Careful definition of V&V strategy: “right property/right tool”
- Deadlock freedom: intrinsic property of Ravenscar
- Configuration: static property with RESOLUTE (model), SPARK (code)
- End-to-end latency?
- Scheduling analysis?
- Safety Analysis?
Flow latency analysis – OSATE

Latency analysis with preference settings: asynchronous system/partition end/worst case as max compute execution time/best case

Latency results for end-to-end flow 'f_6eff' of system 'Crazyflie_Functional_Chain.impl'

Result | Min Specified | Min Actual | Min Method | Max
---|---|---|---|---
abstract Gyro | 0.0ms | 0.1ms | first sampling | 0.3ms
connection Gyro.Gyro_Out -> Fusion.Gyro_In | 0.1ms | 0.0ms | no latency | sampling
abstract Fusion | 0.0ms | 0.0ms | sampling | specified
abstract Controller | 0.2ms | 0.2ms | sampling | 0.4ms
connection Fusion_Data_F_Out -> Controller.Data_F_In | 0.0ms | 0.0ms | no latency | specified
abstract Motors | 0.0ms | 0.0ms | sampling | 0.5ms
connection Controller.Motor_Out -> Motors.Motor_In | 0.1ms | 0.1ms | no latency | 1.6ms
Latency Total | 0.6ms | 0.6ms | specified | 1.6ms
Specified End To End Latency | 0.0ms
End to end Latency Summary

SUCCESS | Minimum actual latency total 0.600ms is greater or equal to expected min
ERROR | Maximum actual latency total 9.60ms exceeds expected maximum end to end latency
WARNING | Jitter of actual latency total 0.600..9.60ms exceeds expected end to end
Scheduling Analysis – AADLInspector / Cheddar
FHA and FMEA – OSATE

Error Source origin

- Used in FHA as “Error”
- propagated type -> hazard
Fault-Tree Analysis Support – OSATE

Use of composite error behavior
- FTA nodes

Use of component error behavior
- Incoming error events + combination

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

component error behavior

transitions

\( t_1 : \text{Operational} - [\text{DOFs\{ValueErroneous\} or processor\{Lost\}} \rightarrow \text{Failed}; \)

propagations

Failed -[]-> \text{Rate}_1\{\text{ValueErroneous};\)
end component;
Using AADL – Conclusion

1. Modeling Architectures: **AADL**
2. Architectural pattern enforcement: **Resolute**
3. Code generation and middleware: **Ocarina** and **PolyORB-HI**
4. V&V strategy
   a) Code level: **SPARK2014**
   b) Model-level: **OSATE ecosystem**
   c) Model checking (not shown)
5. Systems Engineering: **SysML and Capella** (see Adventium Labs talk)
Conclusion

Just an overview of AADL capabilities.

To learn more or collaborate, contact us at info@sei.cmu.edu.

More resources

SEI AADL Library (all papers, technical reports, etc.):
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=453645

SEI AADL resources:
https://www.sei.cmu.edu/our-work/projects/display.cfm?customel_datapageid_4050=191439