SysMLv2 as a DSML to support AADLv2 Semantics and Analysis

Jerome Hugues

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

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.

DM23-0520

BLUF: Extending MBSE with SysML for complex embedded systems architecture

SysML supports systems engineering practice through modeling and model processing.

SysmL1.7 recently released, future SysMLv2 entering finalization

SAE AADL was started at the request of U.S. Army to address complexity in integrating avionics subsystems through a modeling language and a large class of analysis capabilities.

Recurring concern: Do we need two separate languages?

Working with both impacts cost, model update cycles, etc.

A way forward: the best of both worlds

- SysMLv2 provides a foundation to host the AADL language.
- SysMLv2 open API could bring AADL-native V&V capabilities to the SysML world.

Outline

1. **Model-Based for complex avionics systems, the SAE AADL language**
2. SysMLv2 as a host DSML for AADL
3. Leveraging AADL ecosystem

The Safety-Critical Embedded Software System Challenge

Problem:

- Software increasingly dominates safety and mission-critical system development
- Issues discovered long after they are created

Goal:

Early discovery of system-level issues through virtual integration and incremental analytical assurance

Solution:

- **Language** standardized via SAE International & matured into practice through pilot projects & industry initiatives
- **Tooling** available under open source license continually enhances analysis, verification, and generation capabilities
- **Expertise** in Modeling Safety-Critical Embedded Systems

Before You Even Write a Line of Code…

AADL allows you to design the entire system and see where integration 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

AADL backstory

AADL initiated in 1999 after Comanche failure

Goal was to anticipate integration risks for large complex avionics platforms

Note:

- AADL first published in 2004
- SysML in 2006

Evolved in parallel, with different objectives

RAH-66 COMANCHE SOFTWARE REWORK & INTEGRATION COSTS

Photo Credit: Boeing-Sikorsky

Two major software (SW) rebuilds occurred during development indicating significant integration issues

- **1st increment: 75% of SW replaced**
- **2nd increment: 50% of SW replaced**

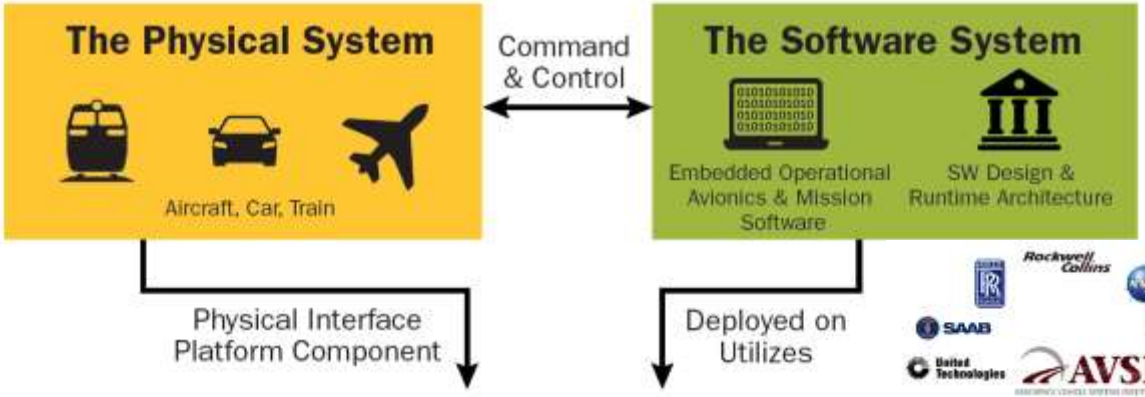
- ***In 1983, the Army planned to buy 5,023 vehicles at $12.1 million/copy.***
- ***Test schedule delays and increasing development costs scaled down the planned buy to 650 aircraft at $58.9 million/copy.***
- ***Most testing involved integration of the complete Mission Equipment Package, which incorporated a radar, infrared, and image-intensified television sensors for night flying and target acquisition.***
- ***Technical challenges remained in software development, integration of mission equipment, radar and infrared signatures, and radar perf.***
- ***The first flight had been originally planned to take place during August 1995, but was delayed by a number of structural and software problems that had been encountered.***
- ***Key program elements, including development and integration of certain software capabilities, failed to foster confidence with Army overseers; several capabilities were viewed as having been unproven and risky.***
- ***The anticipated consumption of up to 40% of the aviation budget by the Comanche alone for a number of years was considered to be extreme.***

References:

- *http://www.defense-aerospace.com/articles-view/release/3/32273/pentagon-hit-over-comanche-failings-(jan.-23).html*
- *https://en.wikipedia.org/wiki/Boeing%E2%80%93Sikorsky_RAH-66_Comanche#cite_note-26*
- *https://en.wikipedia.org/wiki/Boeing%E2%80%93Sikorsky_RAH-66_Comanche#cite_note-Eden_p139-9)*

Comanche costs were expected to consume up to 40% of US Army Aviation budget resulting in cancellation. Integration and software rework were significant cost contributors.

Architecture Analysis & Design Language (AADL) Standard Targets Embedded Software Systems



SAE International

AS 5506 Standard Suite

Standards provide long-term industry-wide solutions to support multi-organization model-based engineering

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

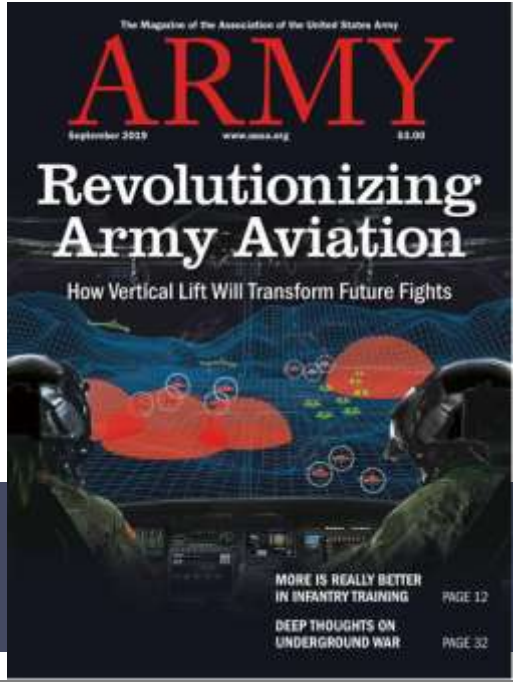
AADL captures mission and safety critical embedded software system architectures in virtually integrated analyzable models to discover system level problems early and construct implementations from verified models

Helping to Revolutionize Army Aviation

Over many years, the SEI has had an outstanding partnership with the US Army, who is at the vanguard of applying AADL and ACVIP to the Army's future vertical lift challenge.

Benefits of AADL & ACVIP (via Alex Boydston)

- Decreased fielding time by finding problems early
- Early risk reduction by discovering performance issues early
- Increased cybersecurity by using AADL/ACVIP to improve system security
- Decreased development costs and support for MOSA and certification by transforming procurement supporting MBE and ACVIP



Virtual integration of software, hardware, and system supports verification, airworthiness, safety, and cybersecurity certification

Impact

Finding Problems Early (AMRDEC/SEI)

Summary: 6 week virtual integration of health monitoring system on CH47 using AADL

Result: Identified 20 major integration issues early

Benefit: Avoided 12-month delay on 24 month program

CH47 Chinook

DARPA

High Assurance Cyber Military Systems (HACMS)

Unmanned Quadcopter

Unmanned Little Bird

TARDEC Autonomous Truck

Improving System Security (DARPA/AFRL)

Summary: AADL applied to unmanned aerial vehicles & autonomous truck

Result: AADL models enforced security policies and were used to auto-build the system

Benefit: Combined with formal methods verification, prevented security intrusion by a red team

---

Transforming procurement (Joint Multi-Role)

Summary: Industry/DoD process demonstration

Result: Pre-integration fault identification

Benefit: 10X reduction integration test cost

Makes complex capabilities possible through Agile analytic and virtual integration of real-time safety and security critical cyber physical embedded systems

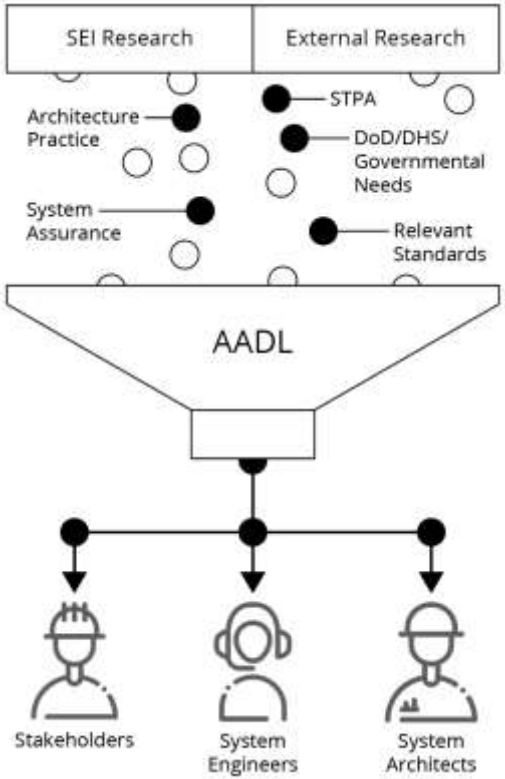
AADL Standard Suite (AS-5506 series)

Core AADL language standard [V1 2004, ... V2.3 2022]

- Safety-critical *embedded system modeling, analysis, and generation*
- Strongly typed language with well-defined semantics for the execution of threads, processes on partitions and processor, sampled/queued communication, modes, and end-to-end flows
- AADL built as a transition vehicle from academic research (e.g., safety, real-time scheduling, code generation, and verification) to industrial practice

Standardized AADL Annex Extensions

- Error Model language for safety, reliability, and security analysis [2015]
- ARINC653 extension for partitioned architectures [2015]
- Behavior Specification Language for modes and interaction behavior [2017]
- Data Modeling extension for interfacing (UML, source code, ASN.1, …) [2011]
- AADL Runtime System & Code Generation [2015]
- FACE Annex [2019]



Outline

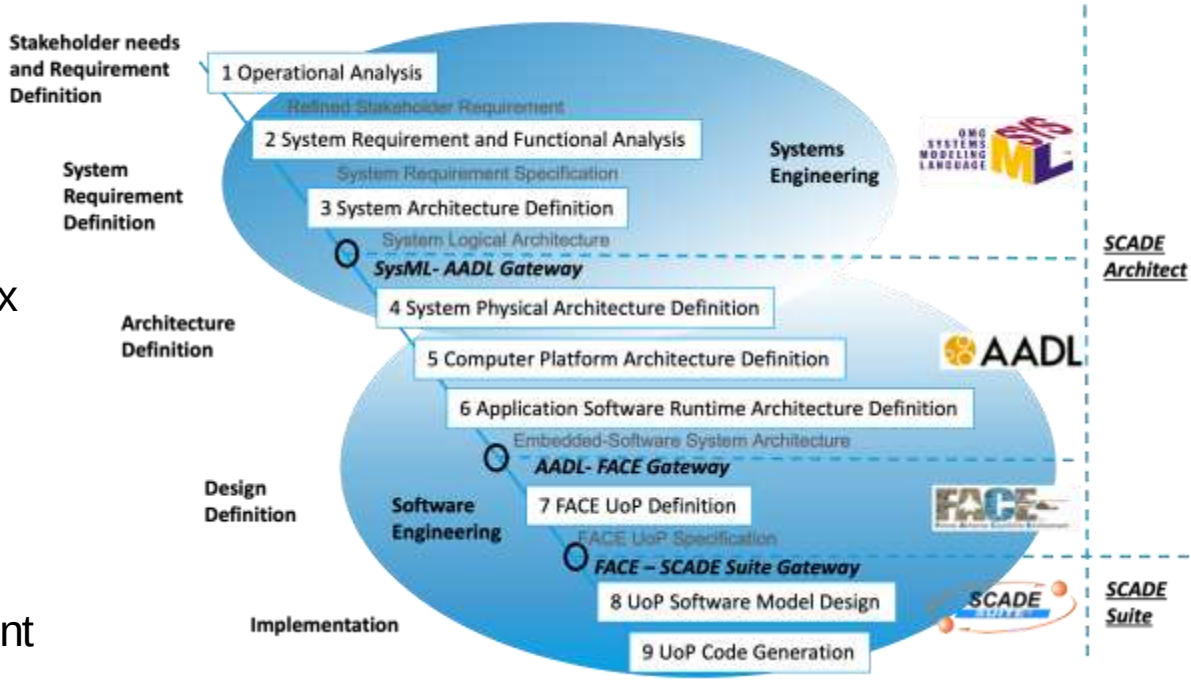
1. Model-Based for complex avionics systems, the SAE AADL language
2. **SysMLv2 as a host DSML for AADL**
3. Leveraging AADL ecosystem

SysMLv1 + AADL workflows are brittle

SysMLv1 to AADL mappings
Brittle, multi-tools,
different representations

Language constructs are heterogeneous, making it complex for designers to navigate the modeling space

Tested in 2018 with ANSYS, technical solution in one tool (SCADE Architect) is more efficient



From: W. Zhe, J. Hugues, J.-C. Chaudemar, and T. LeSergent, "An Integrated Approach to Model Based Engineering with SysML, AADL and FACE," In *Proceedings of Aerospace Systems and Technology Conference 2018 (ASTC'18)*, 2018. 10.4271/2018-01-1942

About SysMLv2

OMG initiated an RFP for SysMLv2 in 2017.

- https://www.omgsysml.org/SysML-2.htm
- Goal is to have a clean start for SysMLv2, without UML legacy
- “Drain the semantics swamp” of SysML1.x

SysMLv2 SST is driving the answer to this RFP, and defines three documents:

- **KerML**: “provides a syntactic and semantic foundation for creating application specific modeling languages.”
- **Systems Modeling Language** (SysML): version 2.0 builds on top of KerML.
- **APIs and services**: to interact with both KerML and SysML.

All drafts and pilot implementation available at https://github.com/Systems-Modeling/.

SysMLv2 / KerML

KerML

KerML defines the foundations for

- Elements and relationships
- Annotations
- **Namespaces**
- **Specialization**
- **Expressions**

Basis for definition (classifier) and usage (feature)

SysMLv2 also has an extensive library for defining its concepts and an API to manipulate models programmatically

SysML

SysMLv2 builds on top of KerML and defines

- ***Models: dependencies between elements, definition and usage, variability***
- ***Attributes: "characteristics of something"***
- Occurrences: definition of a system timeline: specific points in time, intervals, etc.
- ***Structure: items, parts, ports, connections, allocations***
- ***Behavior: actions and states***
- Calculations: expressions and computations
- Requirements: i.e., a constraint satisfied by a "subject"
- Cases: steps required to produce a result regarding a subject: analysis case, verification case, use case
- Viewpoints/views: subset of a model that interests a particular stakeholder

Why Look at SysMLv2 from an AADL Perspective?

AADLv2 defines

- Concepts for representing an architecture: namespace, hierarchy, connection, etc.
- Component categories that specialize these concepts for describing software-intensive systems
- Properties that are typed attributes for configuring component types and instances

→ Notional alignment with SysMLv2 concepts of parts, ports, and attributes

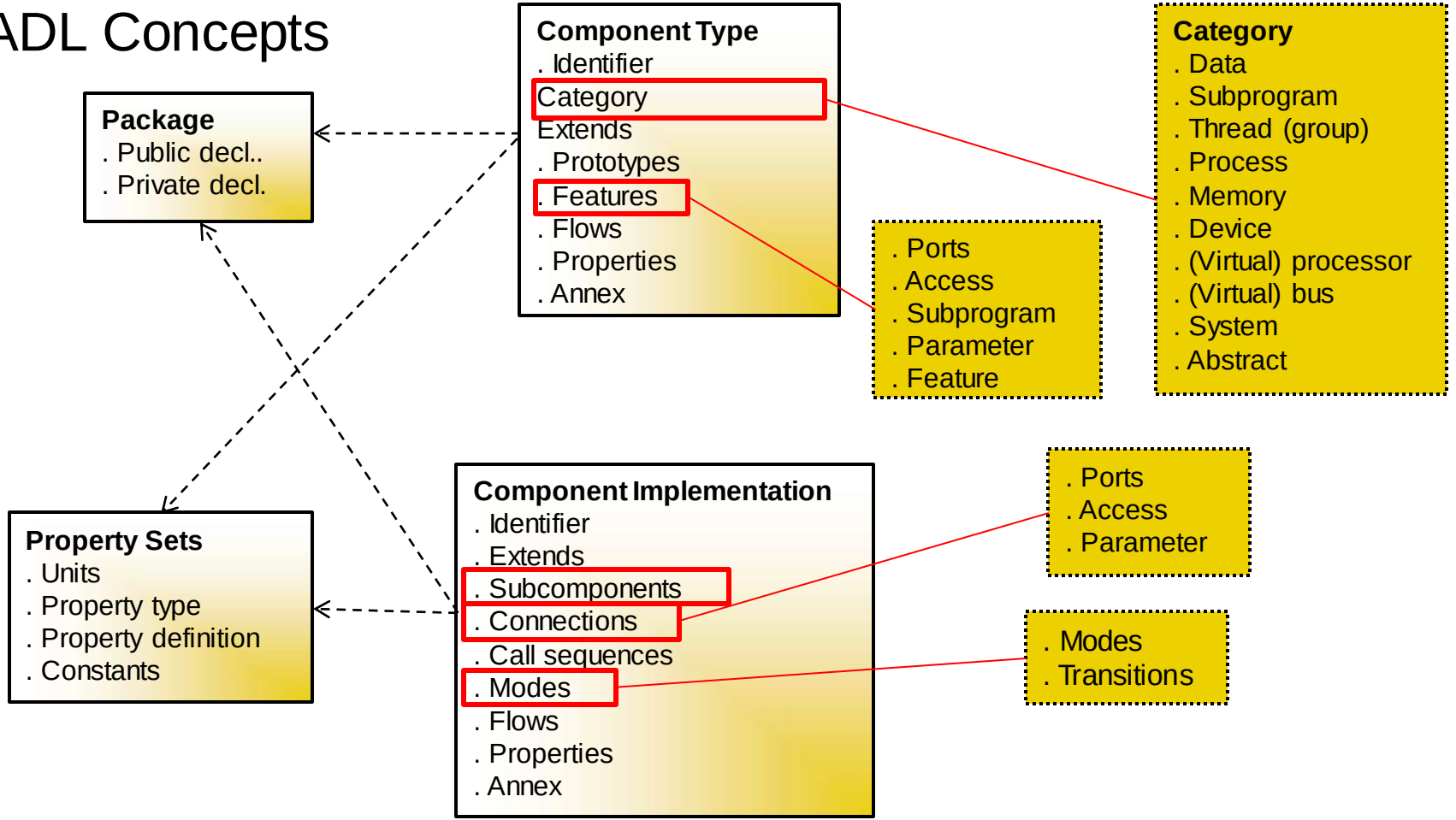
Some AADLv2 extensions are also native in SysMLv2

- SysMLv2 analyses/verification cases and requirements are similar to ALISA (SEI)/Resolute (Collins)
- SysMLv2 behavior has similarities with AADL/BA, SysML1 RAAML with AADL/EMV2 annex

SysMLv2 also has extension mechanisms: new keywords, annex-like constructs similar to AADL

→ How far can we encore AADL semantics in a SysMLv2 library?

AADL Concepts



AADLv2 Native / AADLv2 SysMLv2 Side By Side

Using SysMLv2 type systems to define AADL type hierarchy

```
package AADLv2_Components
public

    /* Component Type -- AADLv2 */
    thread Thread_1 end Thread_1;
    process Process_1 end Process_1;

    /* Component Implementation */
    process implementation Process_1.impl
    subcomponents
       A_Thread_1 : thread Thread_1;
    end Process_1.impl;

end AADLv2_Components;
```

```
package 'AADLv2 Components' {
    import AADL::* ;

    /* Component Type -- AADLv2 */
    part def Thread_1 :> Thread, Type {}
    part def Process_1 :> Process {}

    /* Component Implementation */
    part def Process_1_impl :> Process_1 {
       part A_Thread_1 : Thread_1;
    }

    /* Instance-like approach */
    part def Process_2_impl :> Process {
       part A_Thread : Thread_1;
    }
}
```

Defining an AADL Component Category

In AADLv2 standard, each component category is

- Informally defined as a set of concepts (e.g., process, thread, etc.)
- Formally defined as a set of restriction on features (ports) and subcomponents (parts) from a notional generic component category
- Further specified by a state machine and properties (typed attributes)

Category	Type	Implementation
process	Features: • port • feature group • provides data access • requires data access • provides subprogram access • requires subprogram access • provides subprogram group access • requires subprogram group access • feature Flow specifications: yes Modes: yes Properties: yes	Subcomponents: • data • subprogram • subprogram group • thread • thread group • abstract Subprogram calls: no Connections: yes Flows: yes Modes: yes Properties: yes

AADL Process Category in SysMLv2: Semantics Is an Explicit Artifact

```
part def Process specializes Component {
   attribute redefines category = Component_Category::Process;
   assert constraint {
     checkProcessPorts (portsOnPart) &&
     checkProcessParts (subparts) }}

/* Validity of ports of a process */
constraint checkProcessPorts(p : Port[0..*]) {
   p->forAll { in x : Port ; checkProcessPort(x)}}

constraint checkProcessPort(p : Port) : Boolean[1] {
   p hastype InPort }

/* Validity of parts of a process */
constraint checkProcessParts(p : Part[0..*]) {
   p->forAll { in x : Part ; checkProcessPart(x)}}

constraint checkProcessPart(p : Part) : Boolean[1] {
   p hastype Thread }
```

A Process has a specified category
+ constraints on its parts and ports

Template for constraints on ports
iteration on all ports
+ per-port constraints

Template for constraints on parts
iteration on all parts
+ per-part constraints

Modeling Properties and Property Sets

Property sets as configuration sets, similar to AADLv2 concepts

```
/* Configuration set */
abstract part def Thread_Scheduling_Properties {
        attribute Dispatch_Protocol : Supported_Dispatch_Protocol;
        attribute Priority : Integer;

        /* These properties only applies to Threads. */
        assert constraint { self hastype Thread }
}

part def A_Thread :> Thread, Thread_Scheduling_Properties {
        attribute redefines Priority = 42;
        attribute redefines Dispatch_Protocol =
                Supported_Dispatch_Protocol::Periodic;
}
```

Outline

1. Model-Based for complex avionics systems, the SAE AADL language
2. SysMLv2 as a host DSML for AADL
3. **Leveraging AADL ecosystem**

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

Inline processing: Resolute, BLESS, LAMP
Integration: SysML, FACE, Simulink, SCADE
Architectural pattern checks:
MILS, ARINC, Ravenscar, Synchronous
Model checking:
Timed/Stochastic/Colored Petri Nets
Timed automata et al.: UPPAAL, Versa, TASM
Scheduling: OSATE, CAMET, MAST, Cheddar, CARTS

Performance evaluation: real-time and network calculus
Fault analysis: OSATE, COMPASS,
Mapping to Stochastic Petri Nets, PRISM
Security: CAMET (DoDI 8510.01)
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 and open source toolchains

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

- **OSATE** (SEI/CMU) 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
  - Extensions to OSATE to support other analysis (Multiple Independent Levels of Security (MILS), Framework for Analysis of Schedulability, Timing and Resources (FASTAR)
- **SCADE Architect** (ANSYS Esterel) https://www.ansys.com/products/embedded-software/ansys-scade-suite
  - Eclipse-based tools. Combine SysML, AADL and other formalisms, code generation
- **Stood, AADL Inspector** and **a VS-Code** extension (Ellidiss) https://www.ellidiss.com/
  - Modular toolchain with: graphical and textual editors, model processing/analysis framework
  - Formalized development process, SysML/FACE/Capella import, AADL simulator

Examples of AADL capabilities relevant in a SysMLv2 context

Tool-supported capabilities to enrich the MBSE experience to be ”ported” to SysMLv2

Capability	AADL support – Now	SysMLv2 – Future
Performance	Native language construct, analysis tools	Specific patterns needed, preexisting work in UML MARTE and SPT
Safety	EMV2 annex, FHA and FTA generation, support for STPA approach	SysML1: RAAML diagrams, but no FTA generation SysMLv2: RAAMLv2?, investigate automation
ARINC653	Set of patterns for modeling avionics platforms with precise semantics, verification and code generation	No standard, could build on AADL standard
Integration with OpenGroup FACE	Import FACE UoP as AADL models for analysis	No standard, could build on AADL standard
Assurance plan	ALISA language, + verification methods as user-defined scripts, GSN export	SysMLv2 support verification methods, need patterns for writing assurance plan

The way forward: how to blend SysMLv2 and AADL?

Current library is a proof-of-concept: map AADL syntax to SysMLv2 syntax
⇒ No annex, limited semantics check
⇒ Processing left to third party tools, e.g. translation to AADLv2 textual syntax

Opportunity #1: map AADL behavioral semantics to SysMLv2 execution semantics
⇒ Full system of system analysis capabilities, e.g. simulation, V&V
⇒ Blend with OMG standards for avionics (FACE, ARINC653), safety (RAAML)
⇒ OMG standards are migrating to SysMLv2, continuum from Acquisition to domain-engineering

Opportunity #2: map AADL verification capabilities to SysMLv2 equivalent
⇒ AADL capabilities, e.g. ALISA, Resolute, … have SysMLv2 native equivalent
⇒ SysMLv2 API opens for new workflows: ”ModDevOps” , editor-agnostic V&V, …

Conclusion

SysML and AADL synergies
- Systems engineering applies to safety-critical systems
- SysML has a larger footprint on the system development lifecycle
- Similar language concepts, complementary toolchains, but technical hurdles with SysMLv1

BUT SysML1 does not come with standardized extensions, AADLv2 does

SysMLv2 language capabilities can support AADL concepts
- One language with joint tooling thanks to SysMLv2 API for interoperability

AADL with SysML v2: a unique opportunity for System Engineering to encompass SW subsystems

Opportunities as SysMLv2 finalizes
- Finalize library publication as a published standard
- Enrich SysMLv2 with advanced modeling and analysis capabilities with AADL capabilities
- New verification capabilities as a mid-term objective, using SysMLv2 API