

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

John Hudak, Jerome Hugues

SSD/ACPS/MBE

Software Engineering Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213

Copyright 2021 Carnegie Mellon University.

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](mailto: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

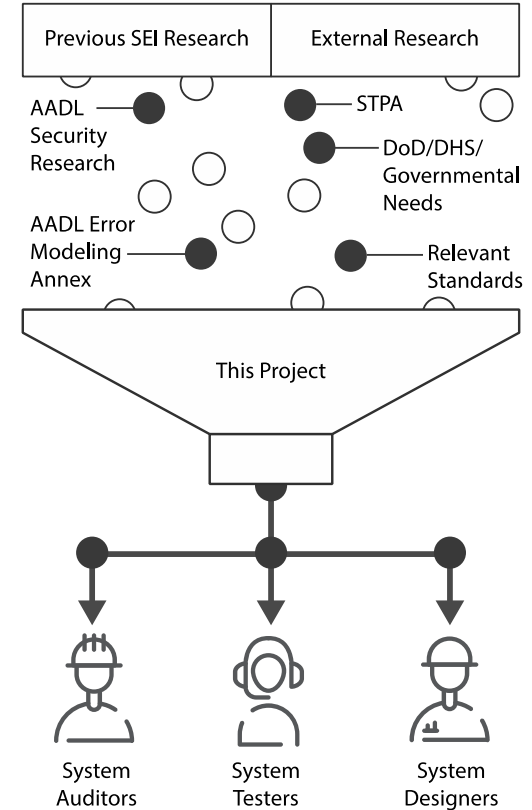
# Making Critical Systems Safer and More Secure

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

## Airbus Gives Alert (Update3)

By Ed Johnson

Oct. 15 (Bloomberg) -- After Australian investigators switched off the jet to nosedive.

The Airbus A330-300 was a computer fed incorrect information. **Australian Transport Safety**

650 feet within seconds, slamming passengers and crew into the cabin ceiling, before the pilots regained control.

"This appears to be a unique event," the bureau said, adding that Toulouse, France-based Airbus, the world's largest maker of commercial aircraft, issued a telex late yesterday 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."

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

Written by [htbw](#) on Oct-7-08 1:48pm  
From: [soyawannaknow.blogspot.com](#)

★★★★☆  Email






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 from Singapore to Perth, Qantas said.

## 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 Buzz:** Hipster's dilemma
-  Boeing & aerospace news
-  Aerospace blog

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.

## Two Crashes In Five Months

### What's Wrong with Boeing's 737 Max 8?

Boeing's new airplane has only been around for two years and already two 737 Max 8s have crashed, killing 346 people. The disasters may be attributable to a design flaw that emerged when engineers began cutting corners.

Boeing's Max 8 is short, limiting ground clearance under the wings. The engine simply doesn't fit.

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.

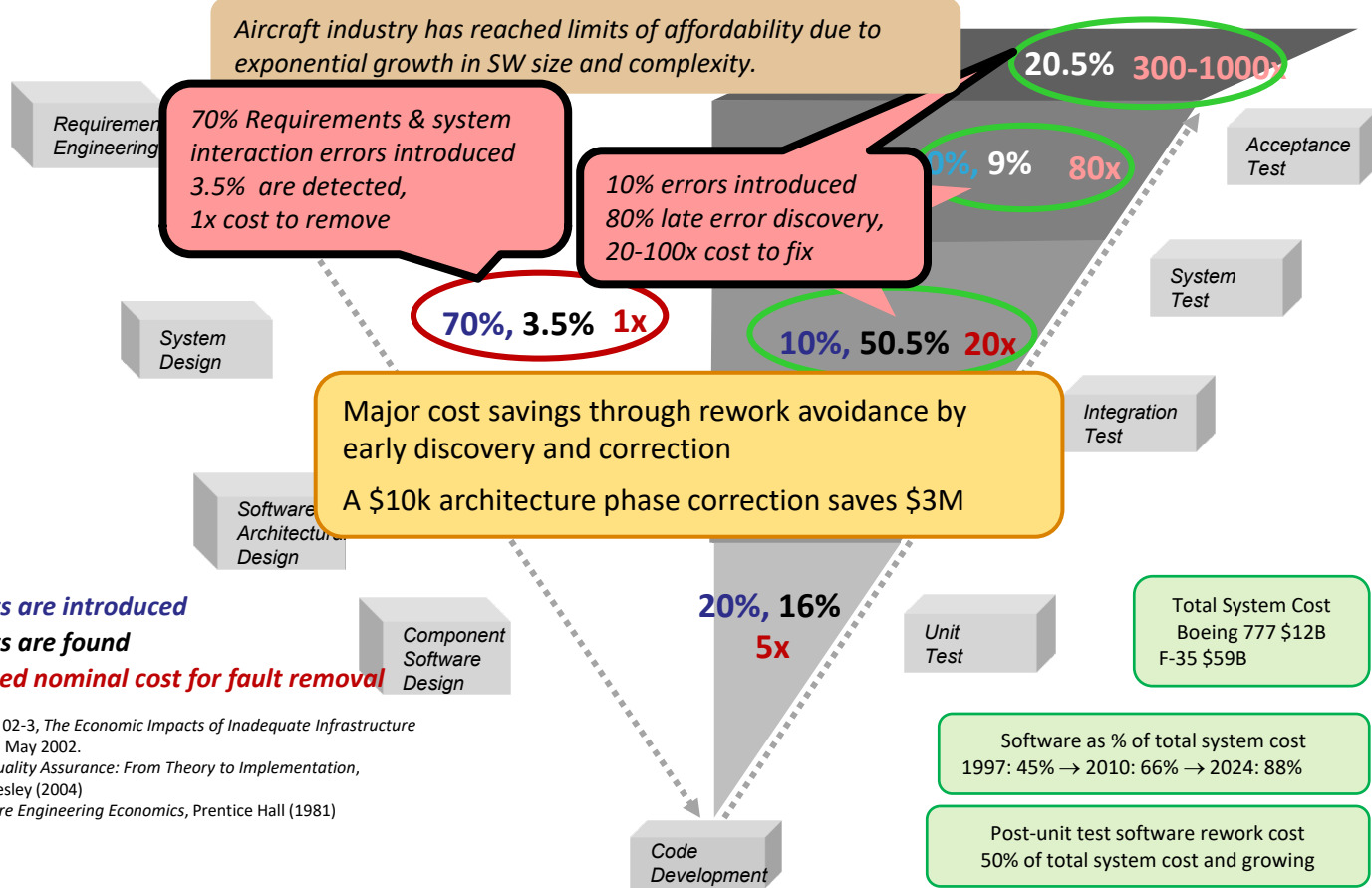
## **Approach:**

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.

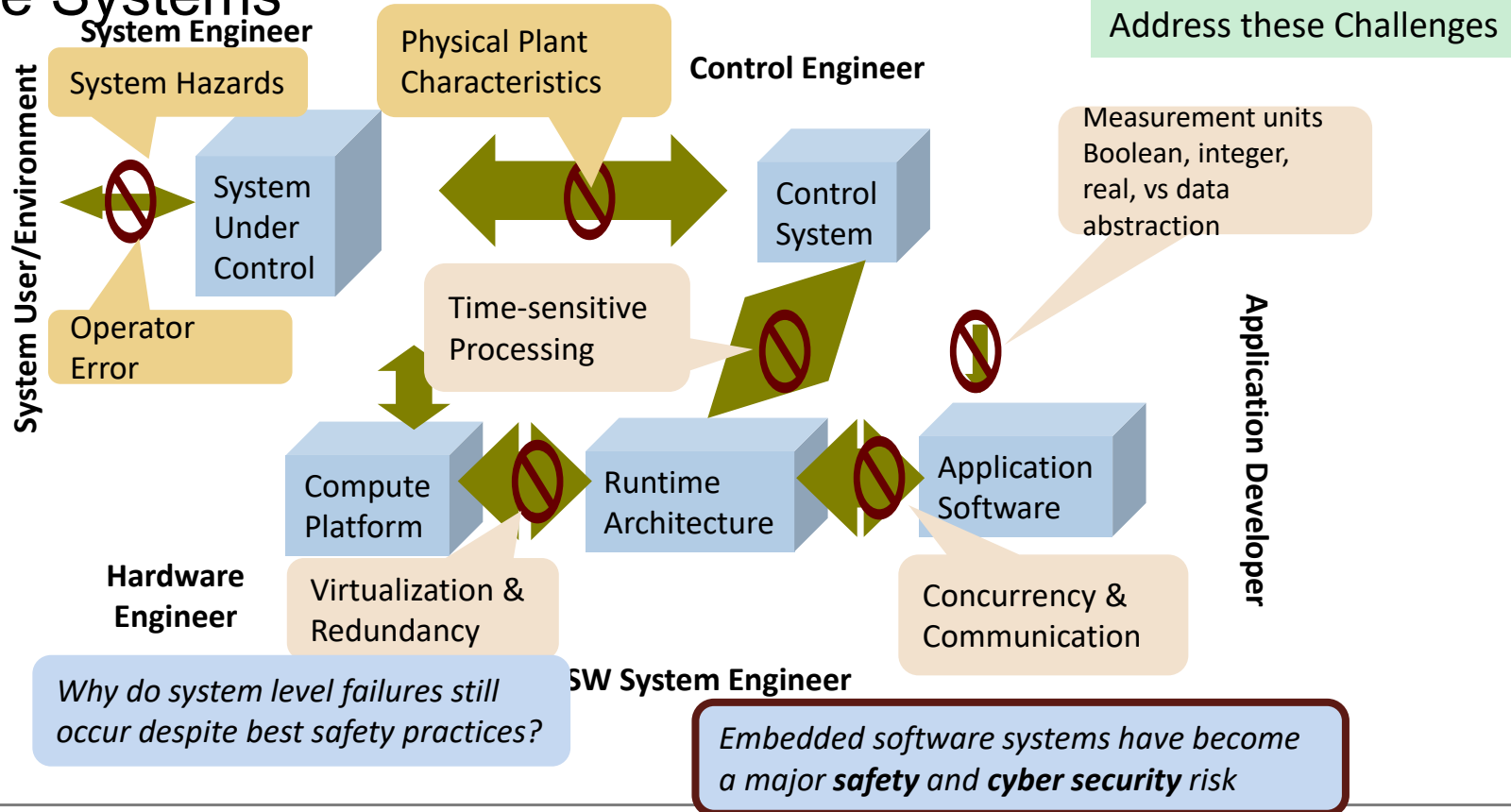
***Reducing Defect Leakage through Early Analytical Assurance is Critical***

# High Fault Leakage Drives Major Increase in Rework Cost

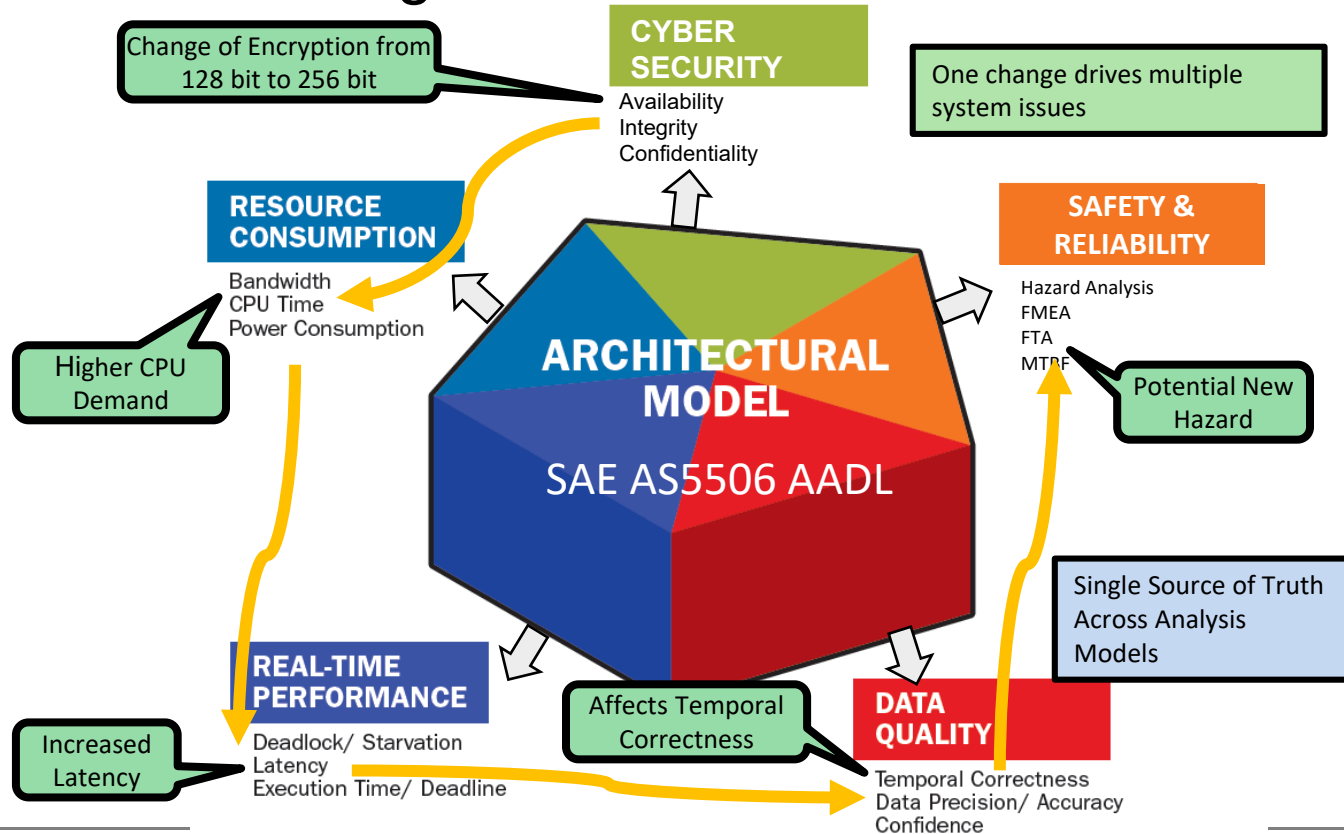


Sources:  
 NIST Planning report 02-3, *The Economic Impacts of Inadequate Infrastructure for Software Testing*, May 2002.  
 D. Galin, *Software Quality Assurance: From Theory to Implementation*, Pearson/Addison-Wesley (2004)  
 B.W. Boehm, *Software Engineering Economics*, Prentice Hall (1981)

# Technical Challenges in Safety-Critical Embedded Software Systems



# Analysis of System Properties via Architecture Model A Contribution to Single Source of Truth



# 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

Concepts

Code



# 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)

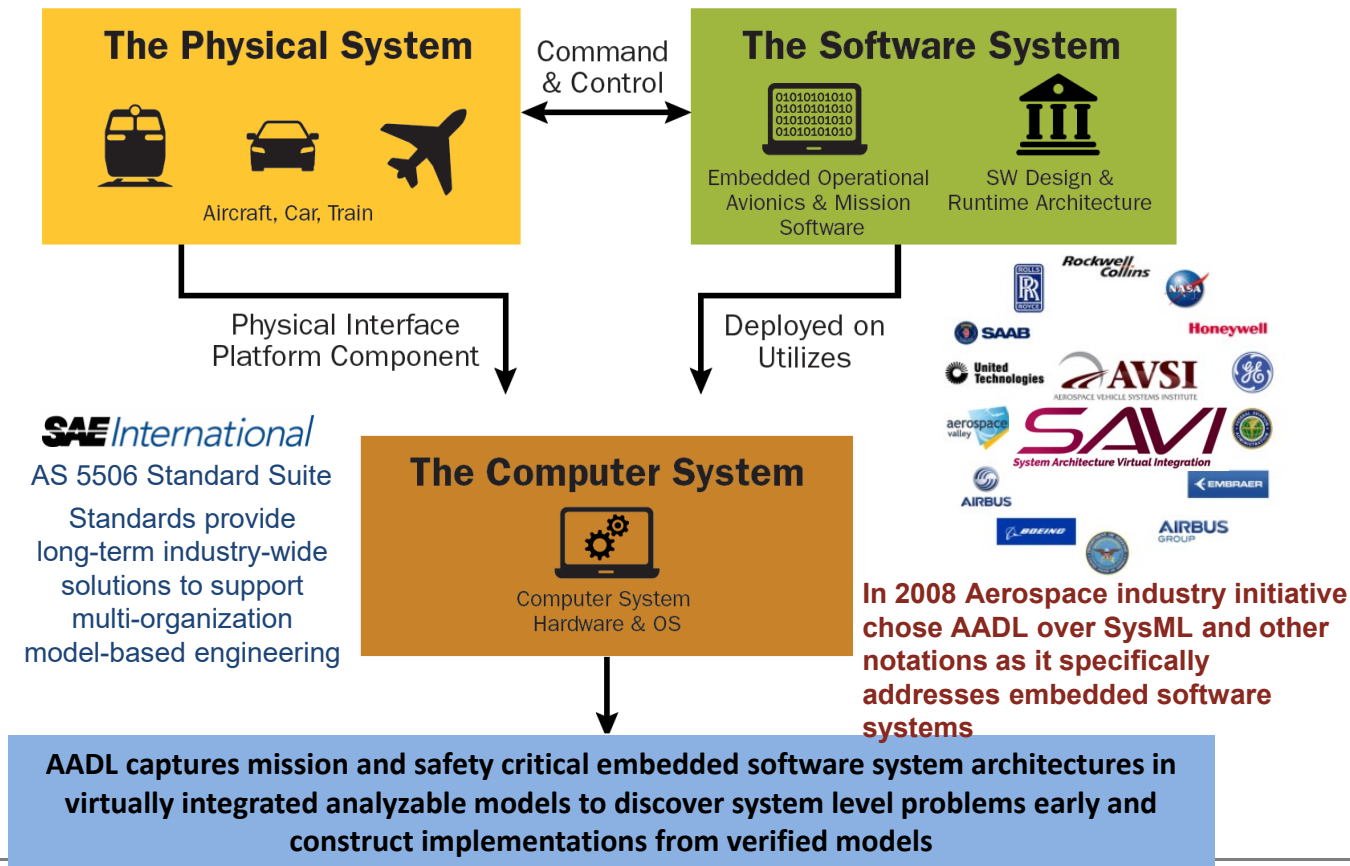
SysML support capturing relationships among system functions, requirements, developers, and users. But not the later development stages

Other modeling notations are required to

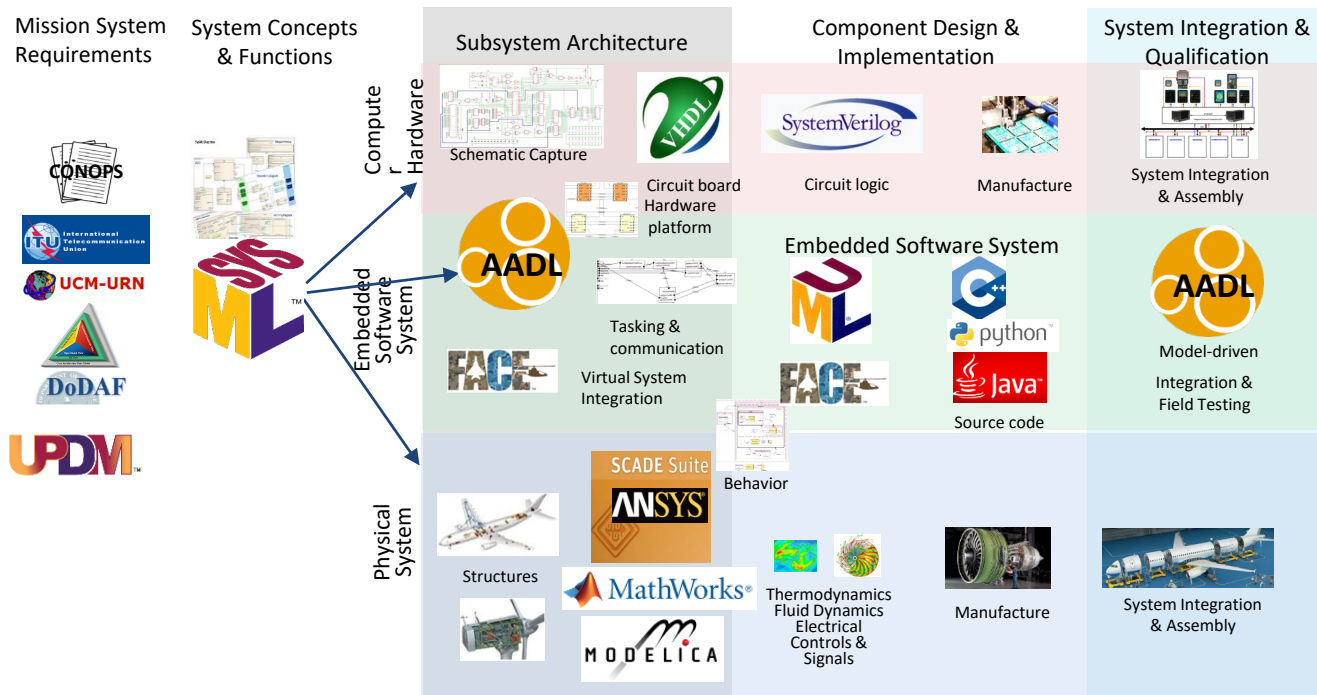
Capture hardware platforms, software architecture, behavioral semantics, deployment of SW to HW, support safety or security assessment, performance analysis, behavioral verification, memory budget validation, etc...



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



# 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

# SAE International AADL Standard Suite (AS-5506 series)

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>

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

## Standardized AADL Annex Extensions

- Error Model language for safety, reliability, security analysis [2006, 2015]
- ARINC653 extension for partitioned architectures [2011, 2015]
- Behavior Specification Language for modes and interaction behavior [2011, 2017]
- Data Modeling extension for interfacing with data models (UML, ASN.1, ...) [2011]
- AADL Runtime System & Code Generation [2006, 2015]
- FACE Annex [2019]

## AADL Annexes in Progress


- Network Specification Annex
- Cyber Security Annex

## Roadmap

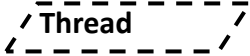
- Requirements Definition and Assurance Annex

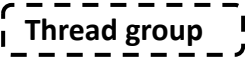
# What are AADL Components?

## Application Components

**System** – hierarchical organization of components 

**Process** – protected address space 

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

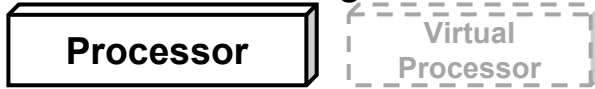
**Thread group** – logical organization of threads 

**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 

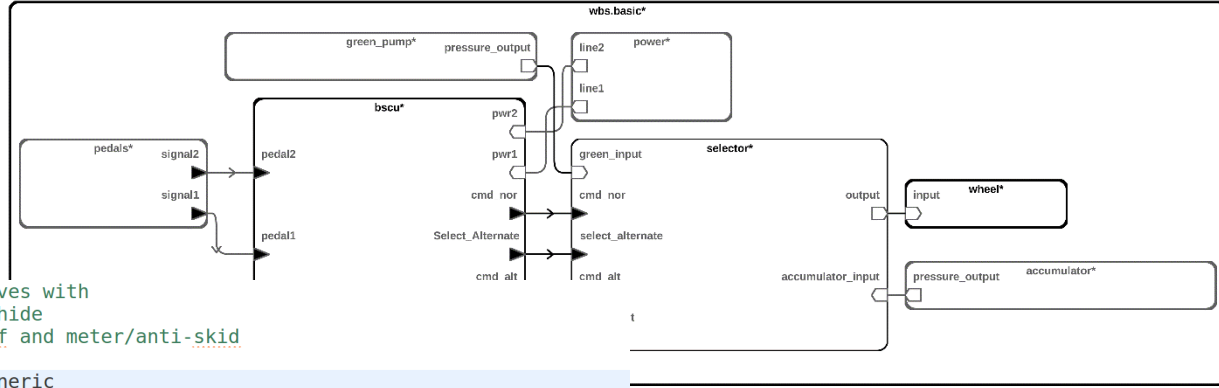
**Device** – interface to external environment



# What does AADL actually look like?

Semi-formal semantics

Only architectural elements



```
79 -- Basic/naive version that abstracts all the valves with
80 -- a selector subsystem. This selector subsystem hide
81 -- the physical logic behind the selector, shutoff and meter/anti-skid
82 -- valves.
83° system implementation wbs.basic extends wbs.generic
84 subcomponents
85   bscu : refined to system impl::bscu::bscu.basic;
86   -- The selector subsystem
87   selector : refined to system impl::valves::selector_basic{Classifier_Substitution Rule => Type_Ext
88   wheel : refined to system impl::wheel::wheel_one_input.i{Classifier_Substitution Rule => Type_Ex
89 connections
90   blue_to_selector : bus access blue_pump.pressure_output <-> selector.blue_input;
91   green_to_selector : bus access green_pump.pressure_output <-> selector.green_input;
92
93   bscu_sel_to_selector : port bscu.Select_Alternate -> selector.select_alternate;
94   bscu_cmdnor_to_selector : port bscu.cmd_nor -> selector.cmd_nor;
95   bscu_cmdalt_to_selector : port bscu.cmd_alt -> selector.cmd_alt;
96
97   selector_to_wheel : bus access selector.output <-> wheel.input;
98 end wbs.basic;
```

Annexes add functionality:

- Error Modeling
- Behavior
- Code Generation

# Textual and Graphical Representation Example

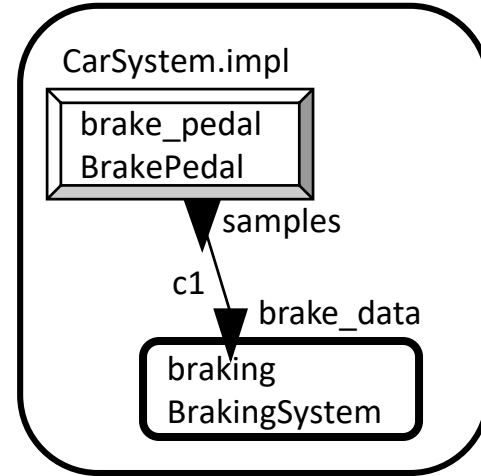
```
system CarSystem
end CarSystem;
```

```
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;
```

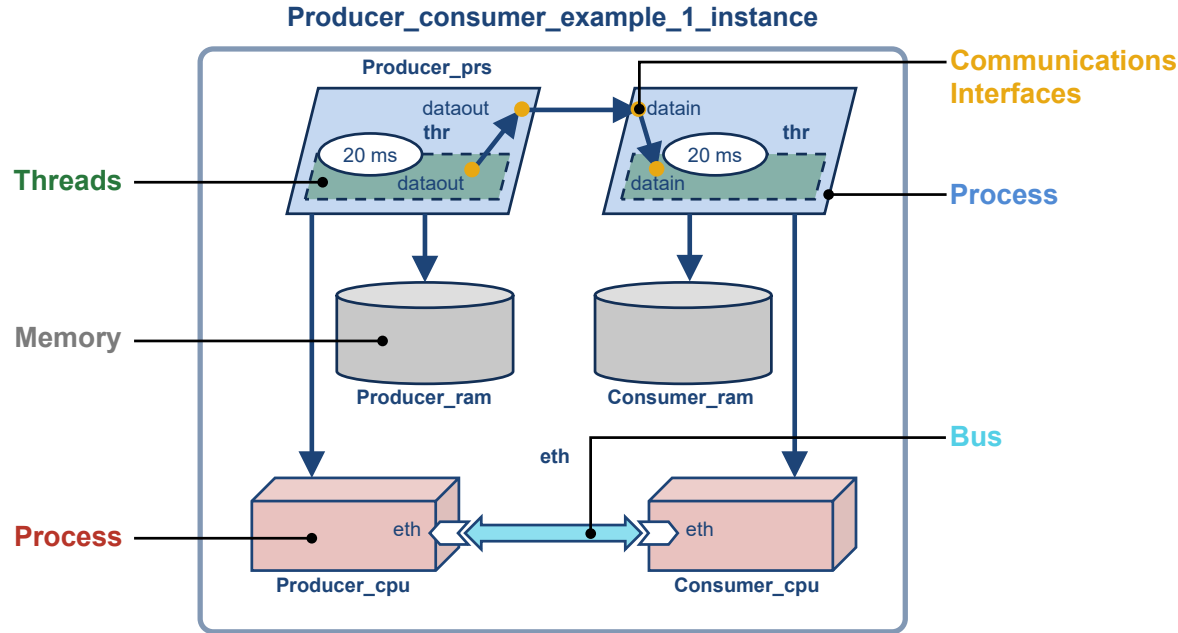
```
system implementation CarSystem.impl
  subcomponents
    braking: system BrakingSystem;
    brake_pedal: device BrakePedal;
  connections
    c1: port brake_pedal.samples -> braking.brake_data;
end CarSystem.impl;
```



What system-level requirements to 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

# An Example Model (graphical)



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).



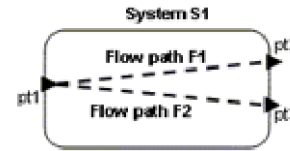
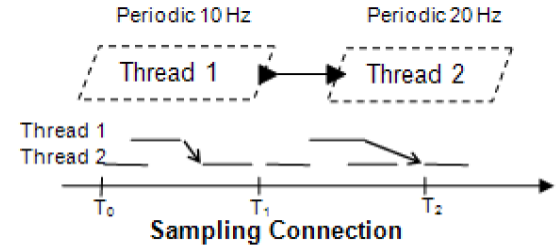
# AADL: Modeling in the small

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 & response processing

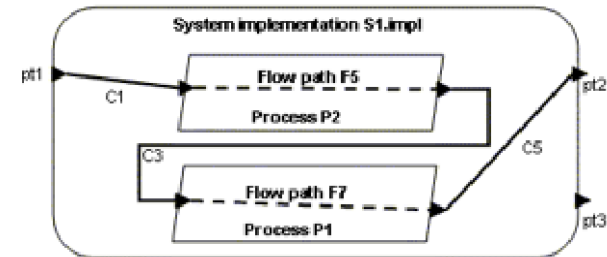
- Data and event **flows**, call/return, shared access
- End-to-End flow specifications



Flow Specification

Flow path F1: pt1  $\rightarrow$  pt2

Flow path F2: pt1  $\rightarrow$  pt3



Flow Implementation of flow path F1

Flow path F1: pt1  $\rightarrow$  C1  $\rightarrow$  P2.F5  $\rightarrow$  C3  $\rightarrow$  P1.F7  $\rightarrow$  C5  $\rightarrow$  pt2

(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/>
  - 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
- **AADL Inspector** (Ellidiss) <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

Analyzable 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

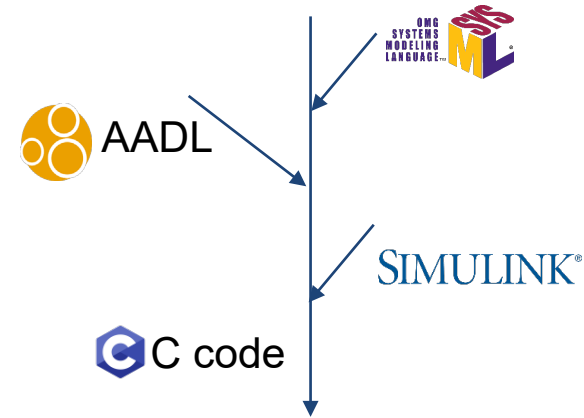
Let's assume we plan at implementing our own UAV control logic, using the Crazyflie as an example

## 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/>

Hardware:

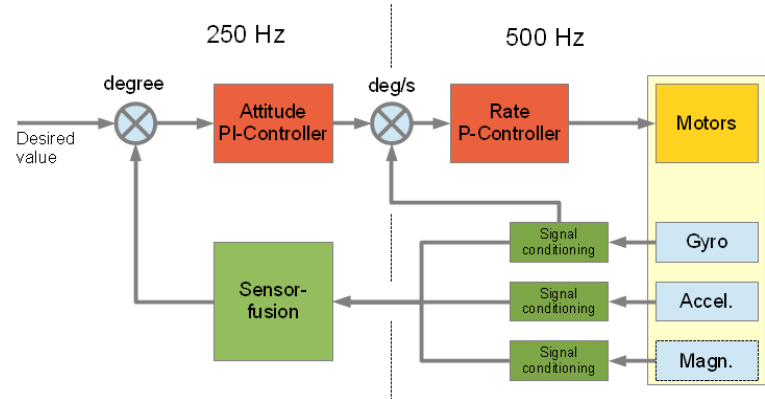
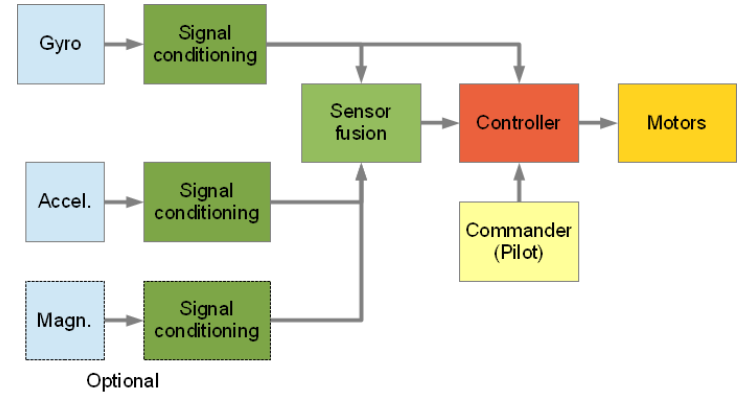
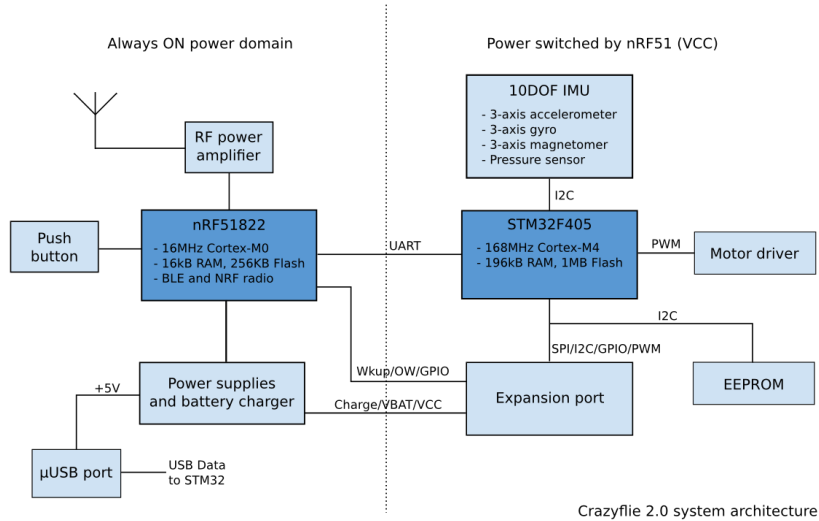
- 2 x MCU : Cortex-M4 + nRF51822
- IMU: MPU-9250 + Pressure Sensor LPS25H
- Bluetooth Low Energy radiocommunication

Software:

- Regular control/command loop
- Manually implemented, in C + FreeRTOS



# Crazyflie architectures

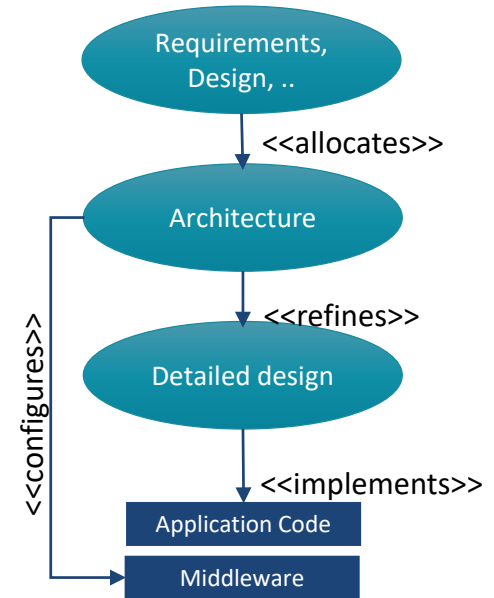


From <https://wiki.bitcraze.io/projects:crazyflie2:architecture:index>

From [https://wiki.bitcraze.io/doc:crazyflie:dev:firmware:sensor\\_to\\_control](https://wiki.bitcraze.io/doc:crazyflie:dev:firmware:sensor_to_control)

# Ideal Systems Engineering Process from models to code

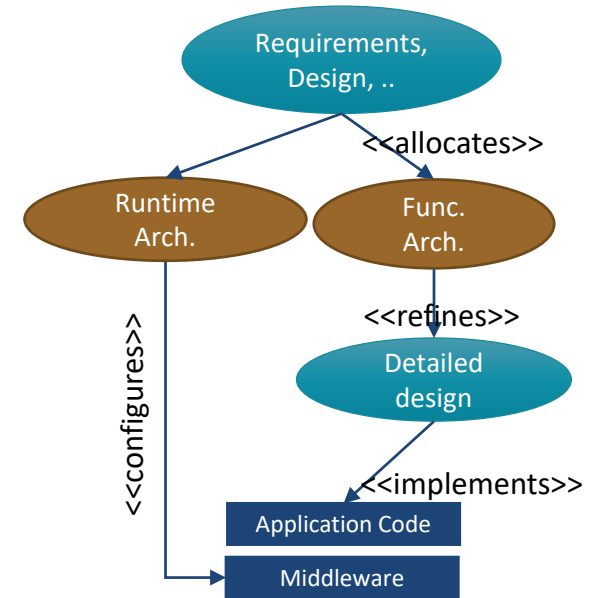
Define a model-based process that



# Ideal Systems Engineering Process from models to code

Define a model-based process that

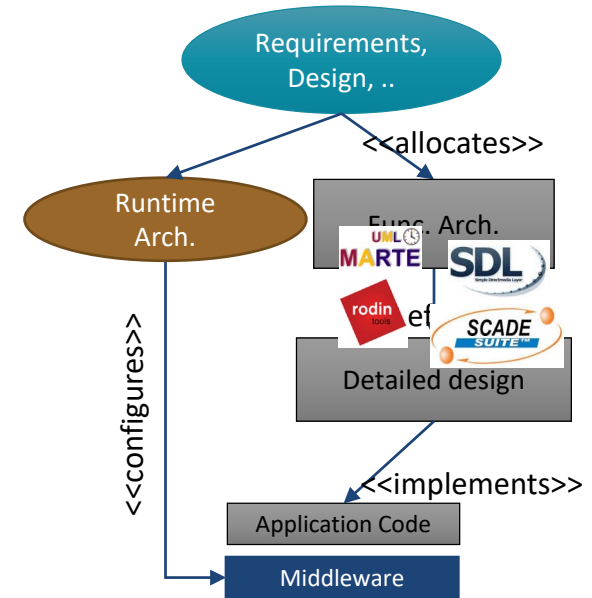
- Separates concerns
  - Runtime architecture ::= configured middleware
  - Functional architecture ::= components



# Ideal Systems Engineering Process from models to code

Define a model-based process that

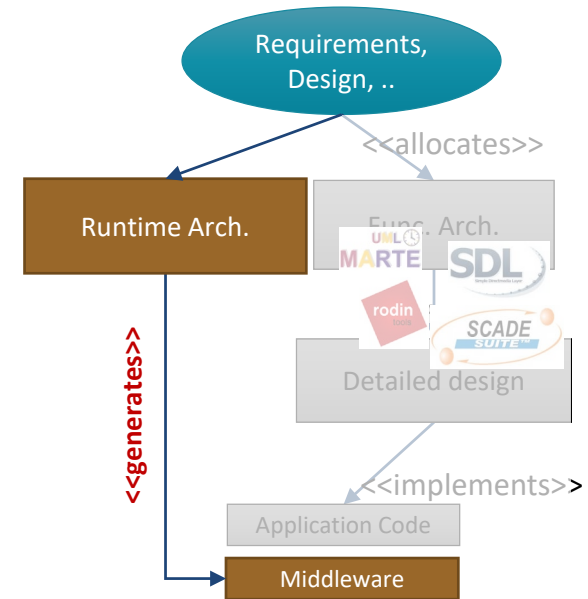
- Separates concerns
  - Runtime architecture ::= configured middleware
  - Functional architecture ::= components
- Mitigates impact of functional code
  - Using safe functional design approach
  - Visible interface and metrics, bounded risks



# Ideal Systems Engineering Process from models to code

Define a model-based process that

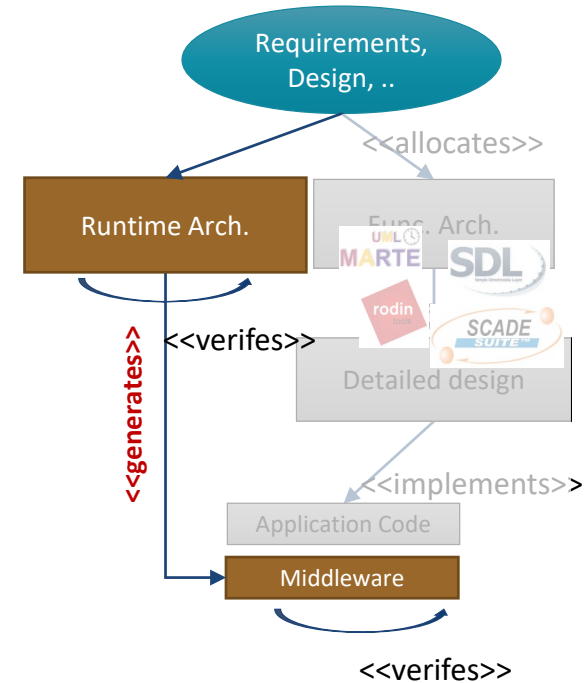
- Separates concerns
  - Runtime architecture ::= configured middleware
  - Functional architecture ::= components
- Mitigates impact of functional code
  - Using safe functional design approach
  - Visible interface and metrics, bounded risks
- Leverages **architectural models** to
  - Generate middleware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



# Ideal Systems Engineering Process from models to code

Define a model-based process that

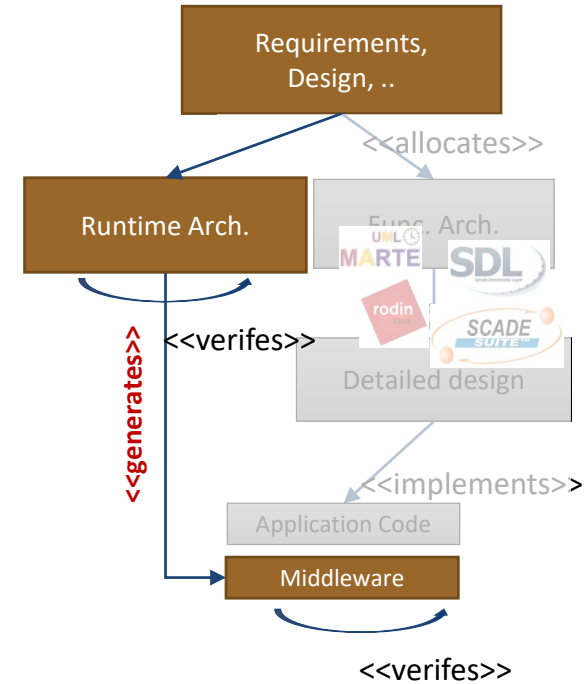
- Separates concerns
  - Runtime architecture ::= configured middleware
  - Functional architecture ::= components
- Mitigates impact of functional code
  - Using safe functional design approach
  - Visible interface and metrics, bounded risks
- Leverages **architectural models** to
  - Generate middleware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



# 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

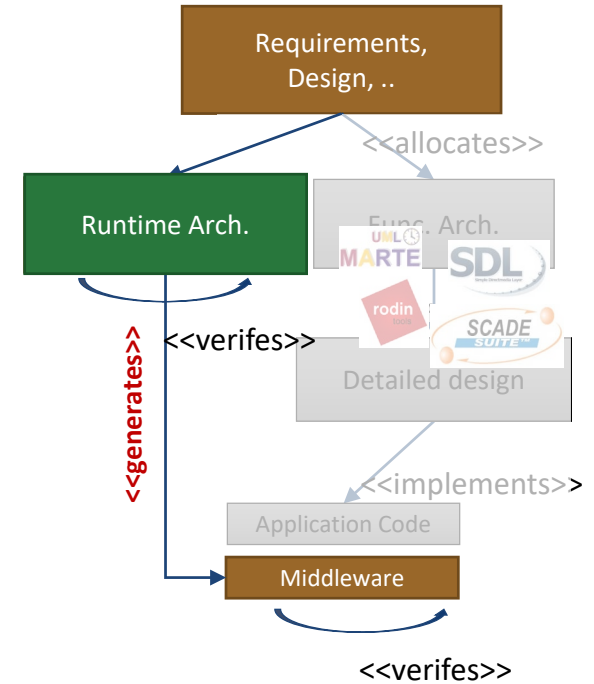


Certification as long term objective



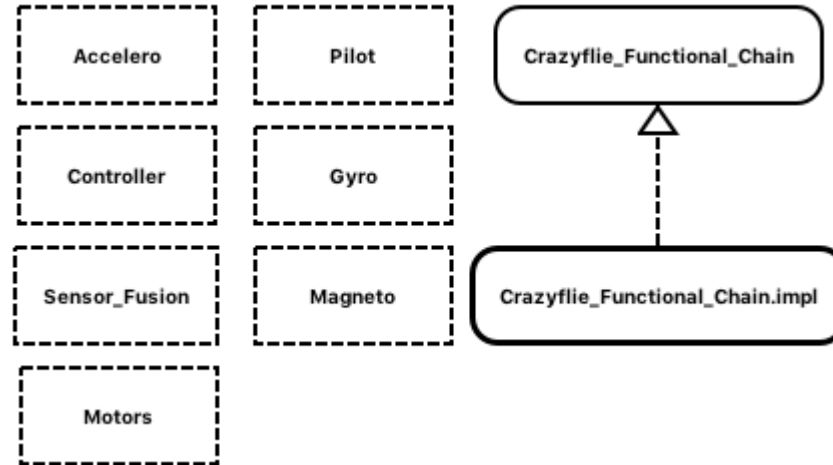
# Using AADL – Outline

## 1. Modeling Architectures

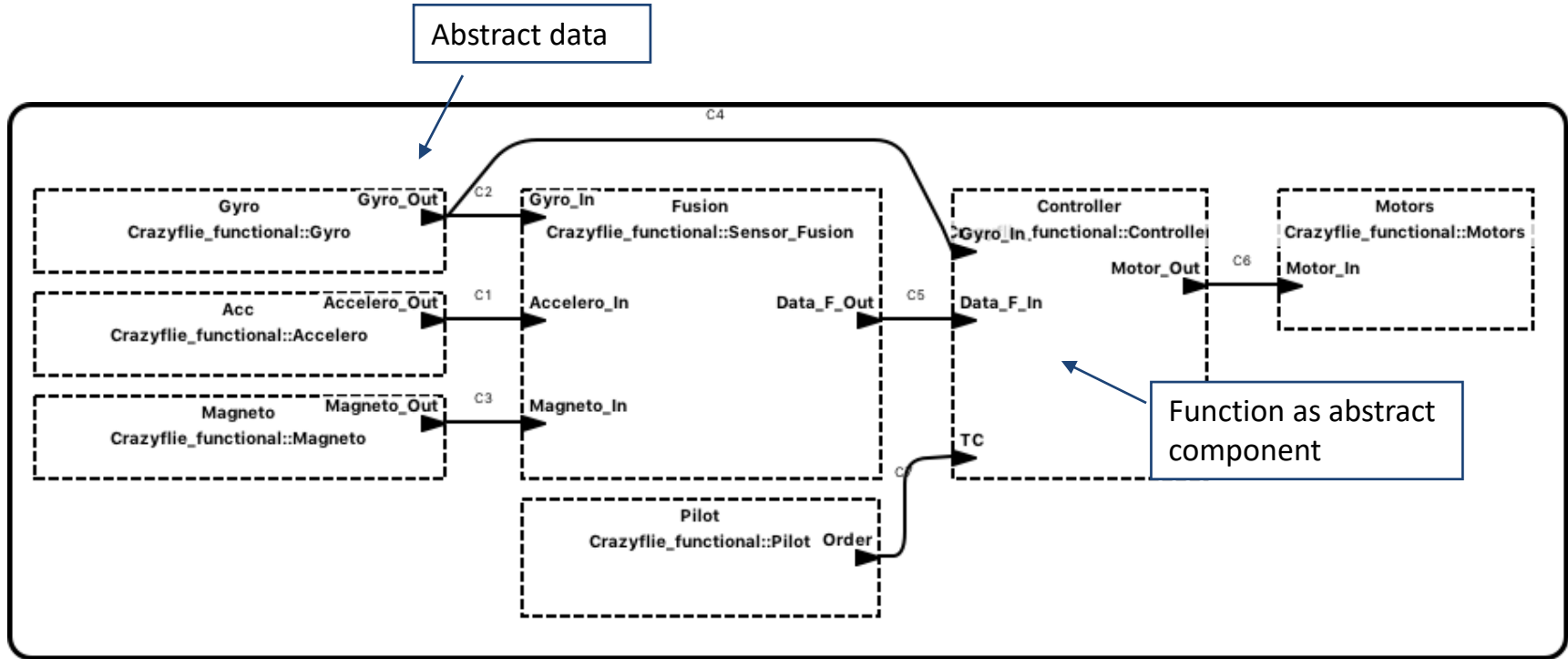


# AADL Functional chain library

Build a package with all functions as abstract components

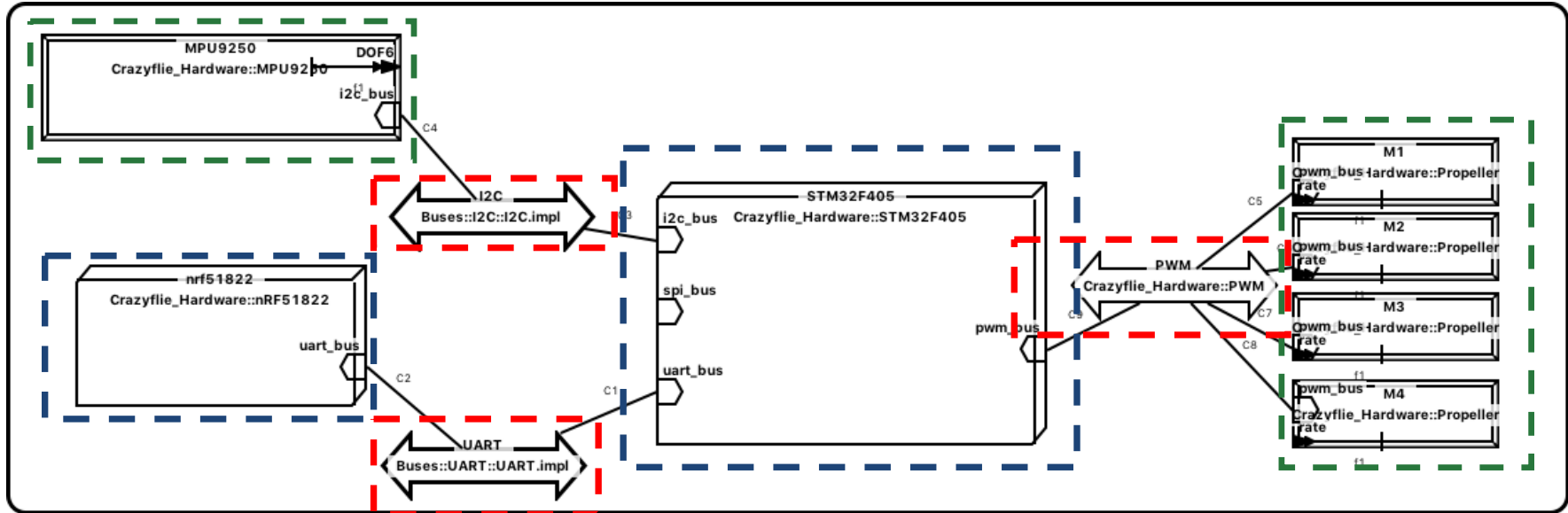


# AADL functional chain implementation



# Crazyflie Hardware

Same strategy: library of components + system implementation

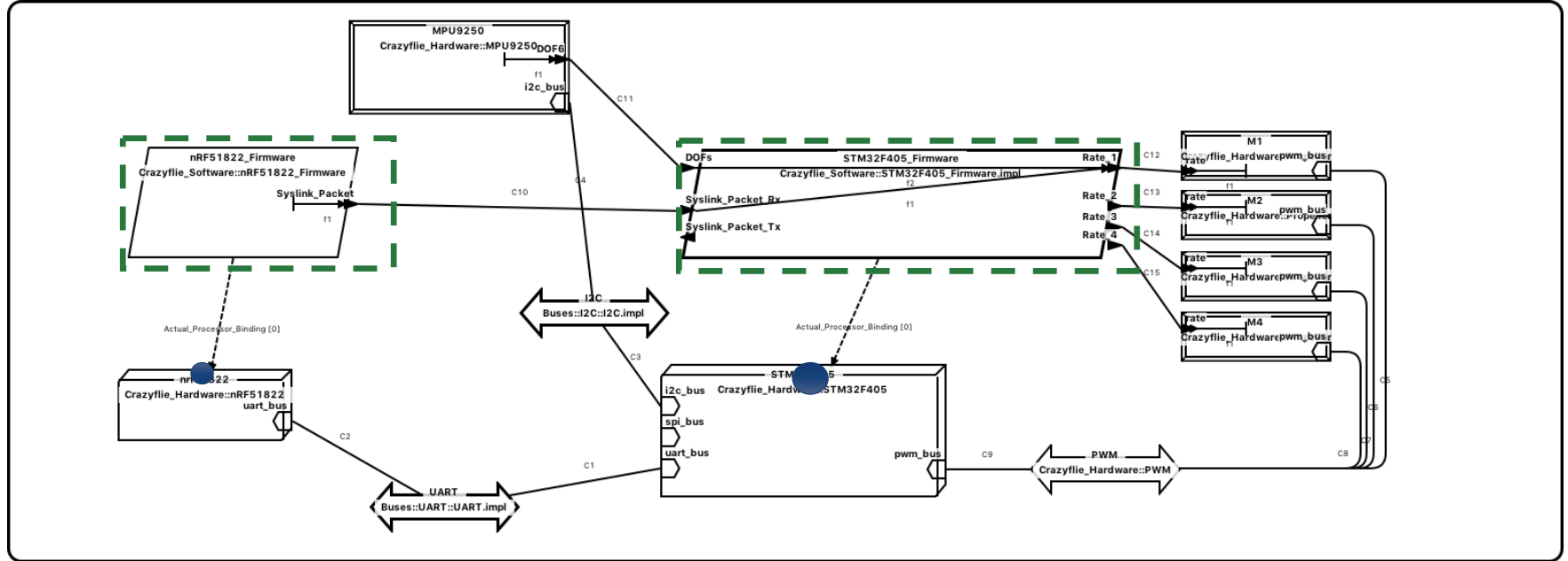


Devices

CPU

Bus components

# Crazyflie Complete System



Software Processes

Hardware/Software Binding

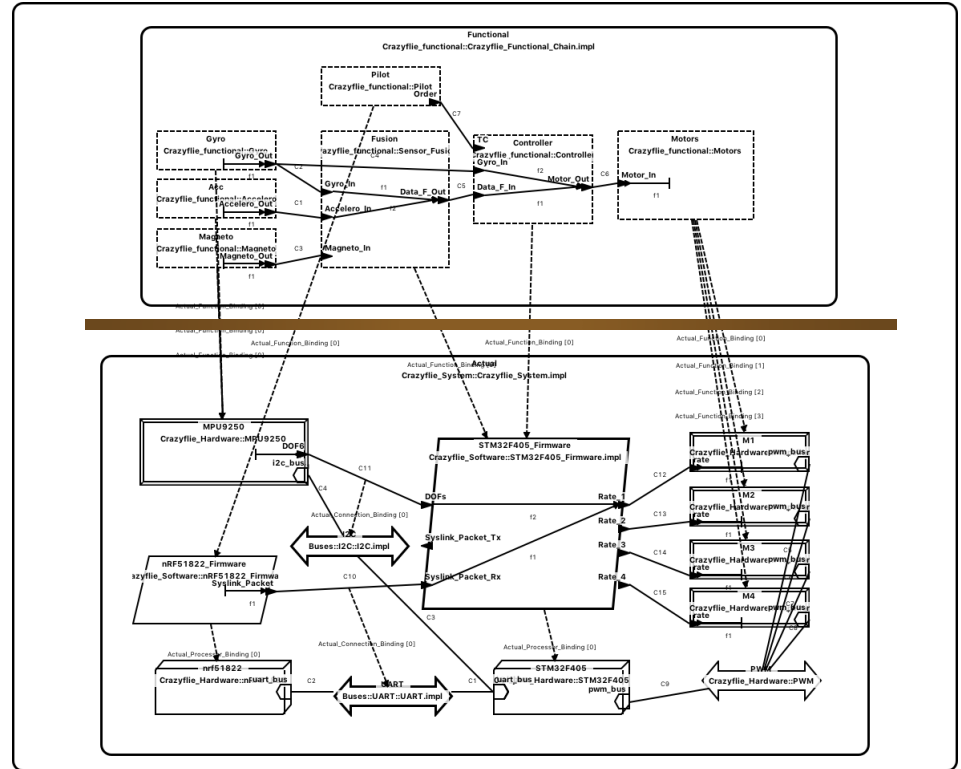
# Functional / Physical bindings (allocation)



Model gateway



Functional



Physical

Code generation



C code

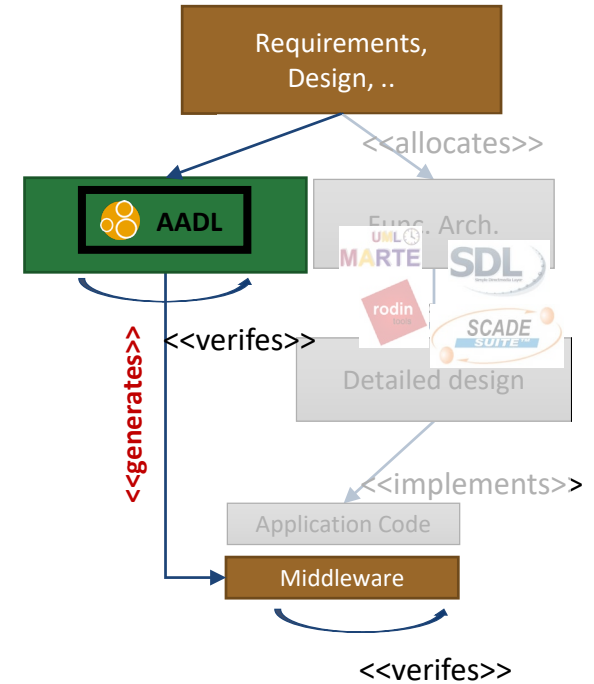


Analysis



# 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

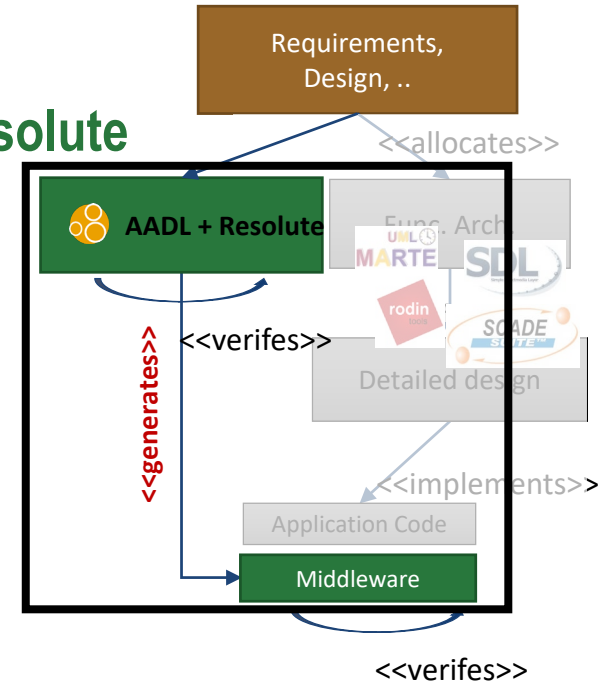
Ravenscar profile for mono-core systems correctly applied

```
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

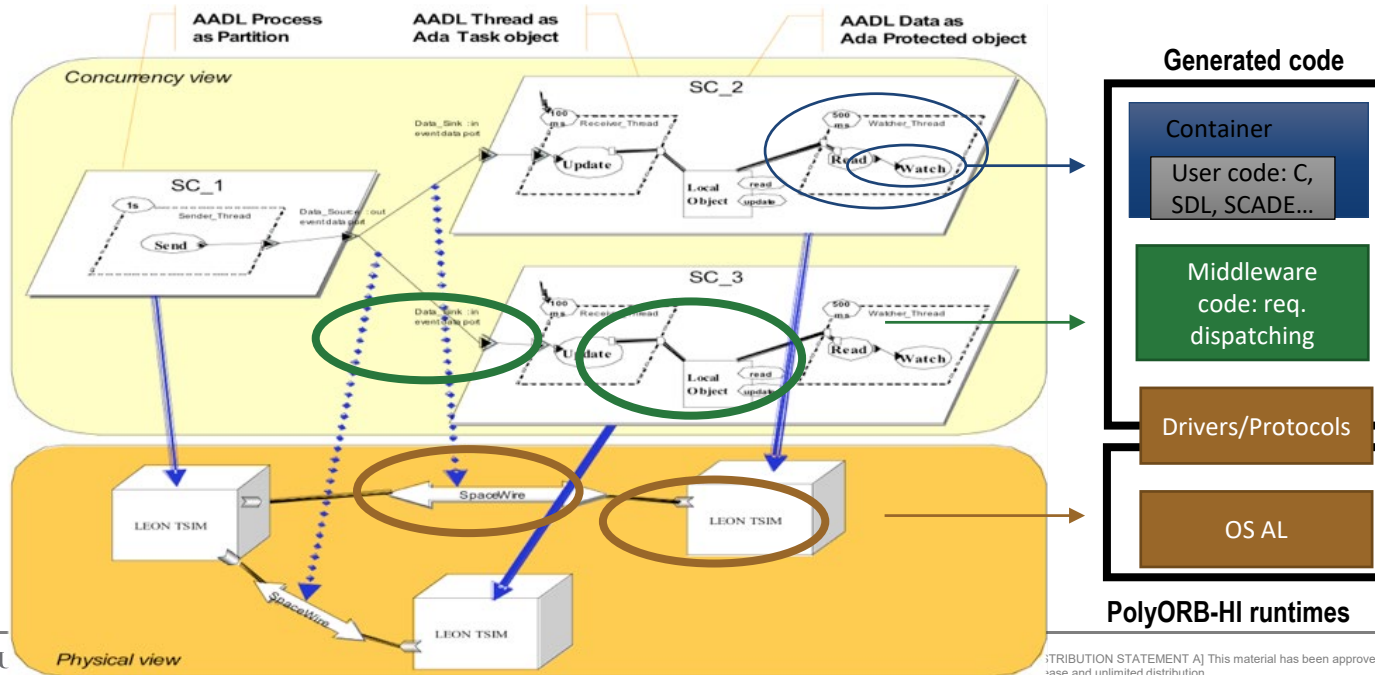
1. Modeling Architectures : **AADL**
2. Architectural pattern enforcement : **Resolute**
3. Code generation and middleware



# 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

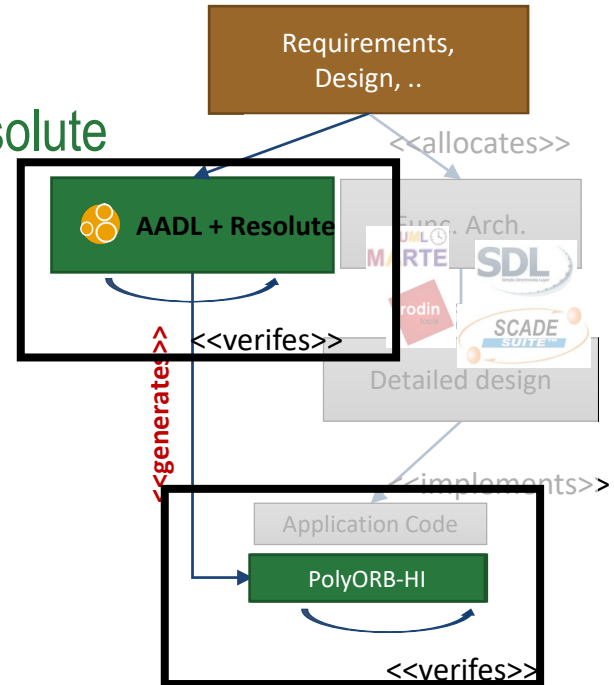
1. Ocarina: AADL model “compiler”, FLOSS
  - Compiler architecture, AADL front-ends, code generation backends
2. PolyORB-HI runtimes
  - Ada High-Integrity profiles, with Ada native and bare board runtimes
  - C POSIX or RTEMS, for RTOS & Embedded,
  - Time and Space partitioning, e.g. ARINC653 C APEX, 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

SPARK Analysis results	Total	Flow	Provers	Justified	Unproved
Data Dependencies	35	32	.	3	.
Flow Dependencies	6	6	.	.	.
Initialization	58	55	.	3	.
Run-time Checks	281	.	281 (CVC4 93%, Trivial 7%)	.	.
Assertions	.	.	.	.	.
Functional Contracts	102	.	63 (CVC4 75%, Trivial 25%)	39	.
Concurrency	14	.	.	14	.
Total	496	93 (19%)	344 (69%)	59 (14%)	.

# V&V Strategy – Model-level

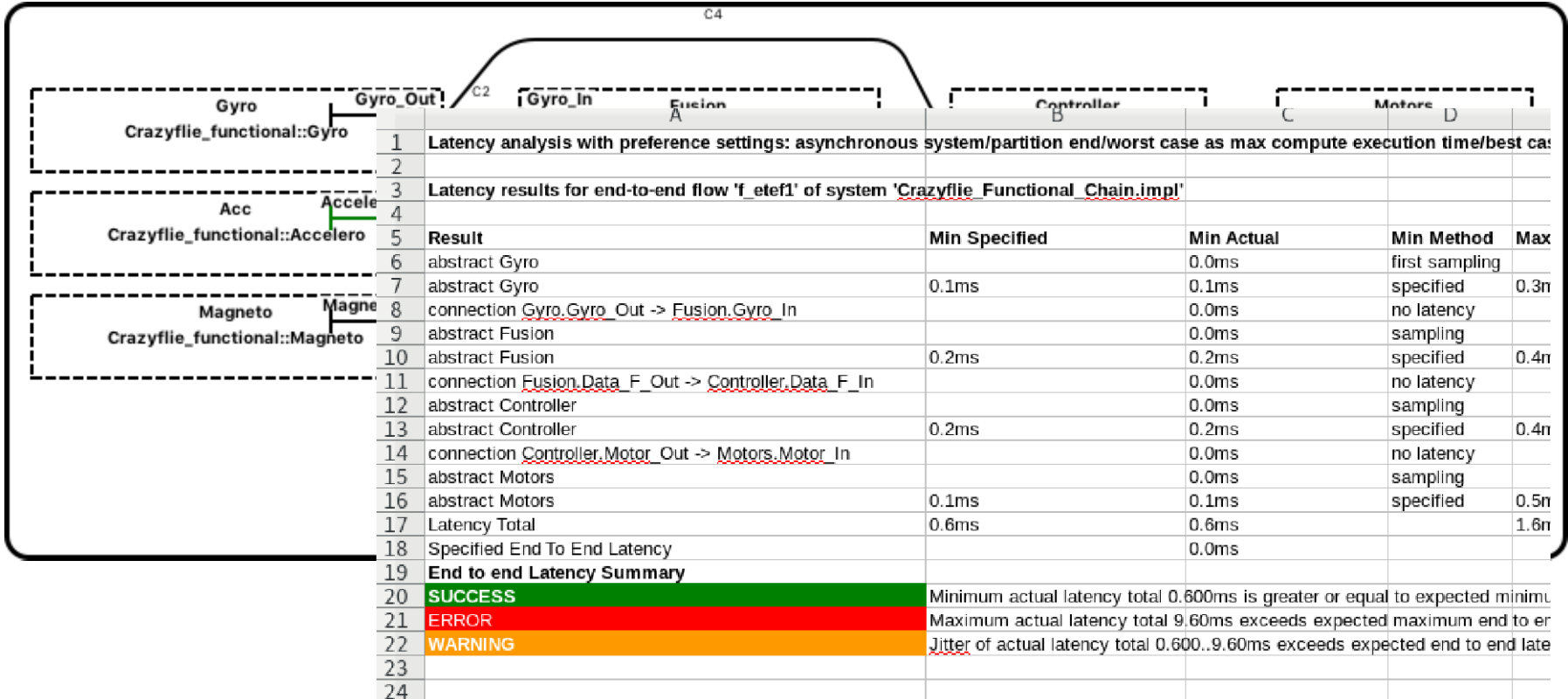
Requirement to assess model-level properties

- 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”

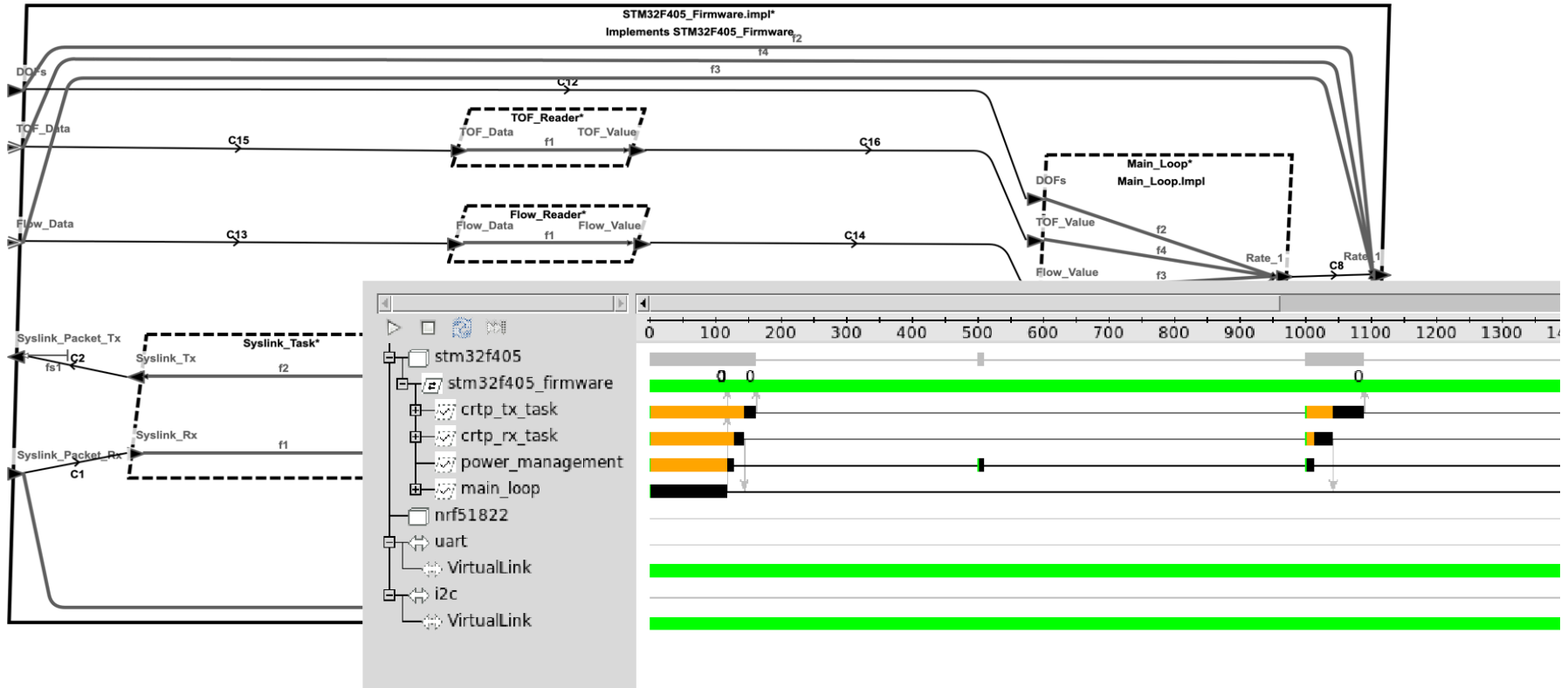
- Deadlock freedom: intrinsic property of Ravenscar
- Configuration: static property with RESOLUTE (model), SPARK (code)
- End-to-end latency ?
- Scheduling analysis?
- Safety Analysis?

# Flow latency analysis – OSATE





# Scheduling Analysis – AADLInspector / Cheddar



# FHA and FMEA – OSATE

## Error Source origin

- Used in FHA as “Error”
- propagated type -> hazard

```

device PositionSensor
  features
    PositionReading: out data port DataDictionary::Position;
  flows
    f1: flow source PositionReading {
      Latency => 2 ms .. 3 ms;
    };
  annex EMV2 {**
    error propagations
      use types ErrorLibrary, FHAErrorLibrary;
      use behavior ErrorModellLibrary::Simple;
      PositionReading: out propagation {ServiceOmission};
      flows

        ef1:error source PositionReading {ServiceOmission} when Failed;
    properties
      EMV2:hazard =>
      [
        crossreference => "1.1.3";
        failure => "Loss of sensor readings";
        phase => "all";
        description => "No stabilator position readings due to sensor failure";
        severity => Critical;
        criticality => Remote;
        comment => "Becomes major hazard, if no reundant sensor";
      ]
      applies to Failed;
    end propagations;
  **};
end PositionSensor;

```

Component	Error	Crossrefe	Functional Failure (Hazard)	Operatio	Environm	Effects of Hazard	Severity	Criticality	Verificati	Comment
PilotCollectiveCommandGr	Failed on DesiredPosition	1.1.9	Grip Failure	all		Pilot cannot provide des	Critical	Remote		Becomes major h
StabilatorPositionSensor	Failed on PositionReading	1.1.3	Loss of sensor readings	all		No stabilator position re	Critical	Remote		Becomes major h

Component	Initial Failure Mode	1st Level Effect	Failure Mode	2nd Level Effect
StabilatorPositionSensor	Failed	StabilatorPositionSensor.PositionReading:{ServiceOmission}	StabilatorPositionSensor.PositionReading-{}->CLTask.Position	StabilatorControl

# 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

```
t1 : Operational -[ DOFs{ValueErroneous} or
processor{Lost}
```

```
] -> Failed;
```

#### propagations

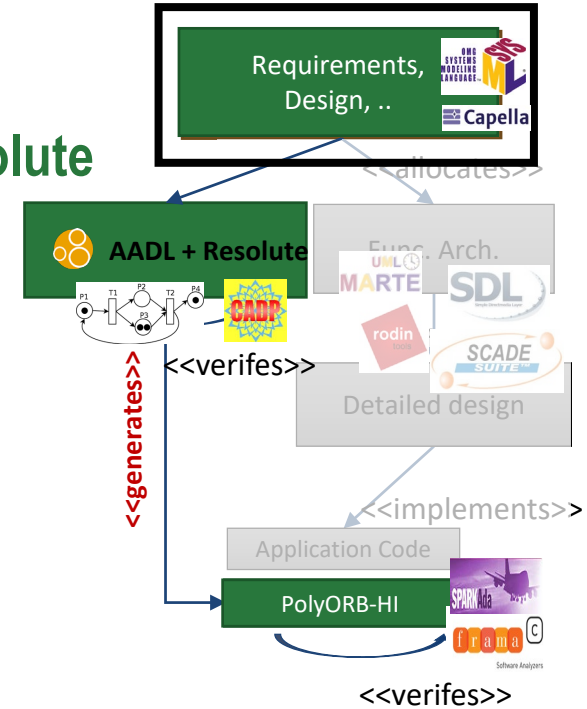
```
Failed -[ ] -> Rate_1{ValueErroneous};
```

```
end component;
```

	Error Model Element/Type	Compu	Specific	Event/Gate Type	Dependent Event
▲ 'Crazyflie_Functional_flow.impl'	error state 'Failed'	4.0e-09		Xor gate	no
▼ 'Crazyflie_Functional_flow.impl'	'Xor' in composite state	3.0e-09		Xor gate	no
▲ 'Crazyflie_Functional_flow.impl'	'Xor' in composite state	2.0e-09		Xor gate	no
▼ 'Crazyflie_Functional_flow.impl'	'Xor' in composite state	2.0e-09		Xor gate	no
○ 'Gyro'	error state 'Failed'		1.0e-09	Basic event	no
○ 'Acc'	error state 'Failed'		1.0e-09	Basic event	no
▼ 'Crazyflie_Functional_flow.impl'	'And' in composite state	1.0e-18		And gate	no
○ 'Magneto'	error state 'Failed'		1.0e-09	Basic event	no
○ 'Optical_Flow'	error state 'Failed'		1.0e-09	Basic event	no
○ 'TOF'	error state 'Failed'		1.0e-09	Basic event	no
▼ 'Fusion'	error state 'Failed'	1.0e-09		Intermediate event	no
▲ 'Fusion'	'Xor' in transition t1	1.0e-09		Xor gate	no
▼ 'Fusion'	'Xor' in transition t1	1.0e-09		Xor gate	no
▲ 'Fusion'	'Xor' in transition t1	1.0e-09		Xor gate	no
▼ 'Fusion'	'Xor' in transition t1	1.0e-09		Xor gate	no
▼ 'Fusion' incoming 'Accelero_In'	{ValueError}			Intermediate event	no
▼ 'Acc' outgoing 'Accelero_Out'	{ValueError}			Intermediate event	no
○ 'Acc' outgoing 'Accelero_Out'	{ValueError} from error source 'ErrorSource'			Basic event	no
▼ 'Fusion' incoming 'Gyro_In'	{ValueError}			Intermediate event	no
▼ 'Gyro' outgoing 'Gyro_Out'	{ValueError}			Intermediate event	no
○ 'Gyro' outgoing 'Gyro_Out'	{ValueError} from error source 'ErrorSource'			Basic event	no
▼ 'Fusion' incoming 'Magneto_In'	{ValueError}	1.0e-09		Intermediate event	no
▼ 'Magneto' outgoing 'Magneto_Out'	{ValueError}		1.0e-09	Intermediate event	no
○ 'Magneto' outgoing 'Magneto_Out'	{ValueError} from error source 'ErrorSource'			Basic event	no
▼ 'Fusion' incoming 'Optical_Flow_In'	{ValueError}			Intermediate event	no
▼ 'Optical_Flow' outgoing 'Optical_Flow_Out'	{ValueError}			Intermediate event	no
○ 'Optical_Flow' outgoing 'Optical_Flow_Out'	{ValueError} from error source 'ErrorSource'			Basic event	no
▼ 'Fusion' incoming 'TOF_In'	{ValueError}			Intermediate event	no
▼ 'TOF' outgoing 'TOF_Out'	{ValueError}			Intermediate event	no
▼ 'TOF' outgoing 'TOF_Out'	{ValueError} from error source 'ErrorSource'			Basic event	no

# 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](mailto: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](https://www.sei.cmu.edu/our-work/projects/display.cfm?customel_datapageid_4050=191439)