Software Engineering Institute | Carn

Modeling System Architectures using the Architecture Analysis and Design Language (AADL)

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Module 1 – Introduction
March 2018

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.

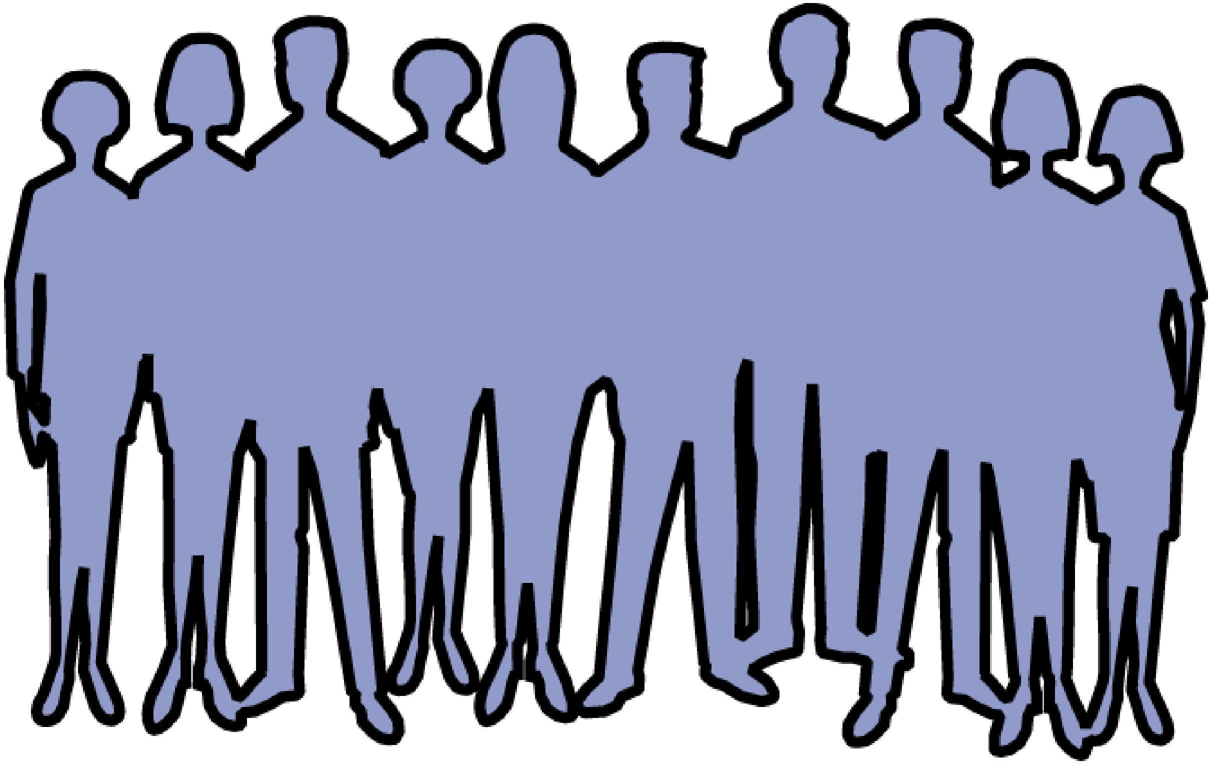
DM18-0221

Introductions

Who are we?

Who are you?

Why are you here?



Objectives for This Course

This course will provide you with:

- an understanding of the value of Architecture-centric Virtual Integration Practice (ACVIP) for system development
- fundamental ACVIP concepts, specifically key principles and methods
- an understanding of software system architecture
- core elements of the Architecture Analysis and Design Language (AADL) modeling language, syntax, semantics, and usage
- modeling and analysis of embedded software systems
- hands on exercises to document and model embedded software system architectures and quantitatively evaluate their quality attributes

The Course Agenda – Days 1-3

Day 1:

- Session 1: Module 1 - AADL Standard & Model-Based Engineering
- Session 2: Module 2 - Conceptualizing a System
- Session 3: Hands-on exercise
- Session 4: Module 3: Modeling and Analyzing Flows

Day 2:

- Session 5: Hands-on exercise
- Session 6: Module 4 - Modeling Software Runtime Characteristics
- Session 7: Hands-on exercise
- Session 8: Module 5- Modeling Execution Platform Components and Devices

Day 3:

- Session 9: : Hands-on exercise
- Session 10: Module 6 - Modeling Logical Resources
- Session 11: Hands-on exercise
- Session 12: Module 8- Modeling Operational modes

The Course Agenda – Days 4-5

Day 4:

- Session 13: Module 8- Hands-on exercise
- Session 14: Module7 & 9- Data modeling, Subprograms, Abstract, Prototypes
- Session 15: Module 2S: Error Modeling and Hazard Analysis
- Session 16: Hands-on exercise

Day 5:

- Session 17: Module 10 - Modeling Guidelines
- Session 18: Modeling discussions/Q&A, topics of interest

Schedule: Day 1

8:30 – 10:15	Introduction and Overview of Modeling and AADL
10:15 – 10:30	BREAK
10:30 – 12:00	Conceptualizing a System
12:00 – 13:00	LUNCH
13:00 – 14:45	Hands-on Exercises
14:45 – 15:00	BREAK
15:00 – 16:30	Modeling and Analyzing Flows

Rules of Engagement

We will be very busy over the next five days. To complete everything and get the most from the course, we will need to follow some rules of engagement:

- Your participation is essential.
- Feel free to ask questions at any time.
- Discussion is good, but we might need to cut some discussions short in the interest of time. (we are happy to discuss topics over lunch, etc.)
- Please try to limit side discussions during the lectures.
- Please turn off your cell phone ringers, refrain from texting.
- Let's try to start on time.
- Participants must be present for all sessions in order to earn a course completion certificate.

Session 1 Objectives

Provide an overview of modeling, software architecture

Introduce architecture-centric virtual integration concepts

Introduce the SAE AADL Standard

Provide a summary of AADL concepts

Introduce a tool strategy for AADL

Outline: AADL Standard & ACVIP

- Challenges in embedded software systems
- Modeling-driven engineering and Architecture-Centric Virtual Integration Practice (ACVIP)
- Overview of SAE AADL Standard suite
- AADL Language Overview
- AADL Tools
- Summary

We Rely on Software for Safe Aircraft Operation

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

Written by htbw on Oct-7-08 1:48pm

From: soyawannaknow.blogspot.com

Thirty-six pas
in a mid-air d
emergency la
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.

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

Oct. 15 (Bloomberg) -- **Airbus SAS** issued an alert to airli
after Australian investigators said a computer fault on a
Ltd. flight switched off the autopilot and generated false
jet to nosedive.

The Airbus A330-300 was cruising at 37,000 feet (11,277
computer fed incorrect information to the flight control sy
Australian Transport Safety Bureau said yesterday. The
650 feet within seconds, slamming passengers and crew
ceiling, before the pilots regained control.

``This appears to be a unique event,'' the bureau said, a
Toulouse, France-based Airbus, the world's largest make
aircraft, issued a telex late yesterday to airlines that fly A
fitted with the same air-data computer. The advisory is `
minimizing the risk in the unlikely event of a similar occurr

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.

Software Problems not just in Aircraft

Expert • Independent • Nonprofit

ConsumerReports.org

This article appeared in May 2010 Consumer Reports Magazine.

Many appliances now rely on electronic controls and operating softw … But it turned out to be a problem for the Kenmore 4027 front-loader, which scored near the bottom in our February 2010 report.

Our tests found that the rinse cycles on some models worked improperly, resulting in an unimpressive cleaning.

When Sears, which sells the washer, saw our February 2010 Ratings (available to subscribers), it worked with LG, which makes the washer, to figure out what was wrong. They quickly determined that a software problem was causing short or missing rinse and wash cycles, affecting wash performance. Sears and LG say they have reprogrammed the software on the models in their warehouses and on about 65 percent of the washers already sold, including the ones we had purchased.

Our retests of the reprogrammed Kenmore 4027 found that the cycles now worked properly, and the machine excelled. It now tops our Ratings (available to subscribers) of more than 50 front-loaders and we've made it a CR Best Buy.

If you own the washer, or a related model such as the Kenmore 4044 or Kenmore Elite 4051 or 4219, you should get a letter from Sears for a free service call. Or you can call 800-733-2299.

May 7, 2010

Lexus GX 460 passes retest; Consumer Reports lifts "Don't Buy" label

Consumer Reports is lifting the Don't Buy: Safety Risk designation from the 2010 Lexus GX 460 SUV after recall work corrected the problem it displayed in one of our emergency handling tests. (See the original report and video: "Don't Buy: Safety Risk--2010 Lexus GX 460.")

We originally experienced the problem in a test that we use to evaluate what's called lift-off oversteer. In this test, as the vehicle is driven through a turn, the driver quickly lifts his foot off the accelerator pedal to see how the vehicle reacts. When we did this with our GX 460, its rear end slid out until the vehicle was almost sideways. Although the GX 460 has electronic stability control, which is designed to prevent a vehicle from sliding, the system wasn't intervening quickly enough to stop the slide. We consider this a safety risk because in a real-world situation this could cause a rear tire to strike a curb or slide off of the pavement, possibly causing the vehicle to roll over. Tall vehicles with a high center of gravity, such as the GX 460, heighten our concern. We are not aware, however, of any reports of injury related to this problem.

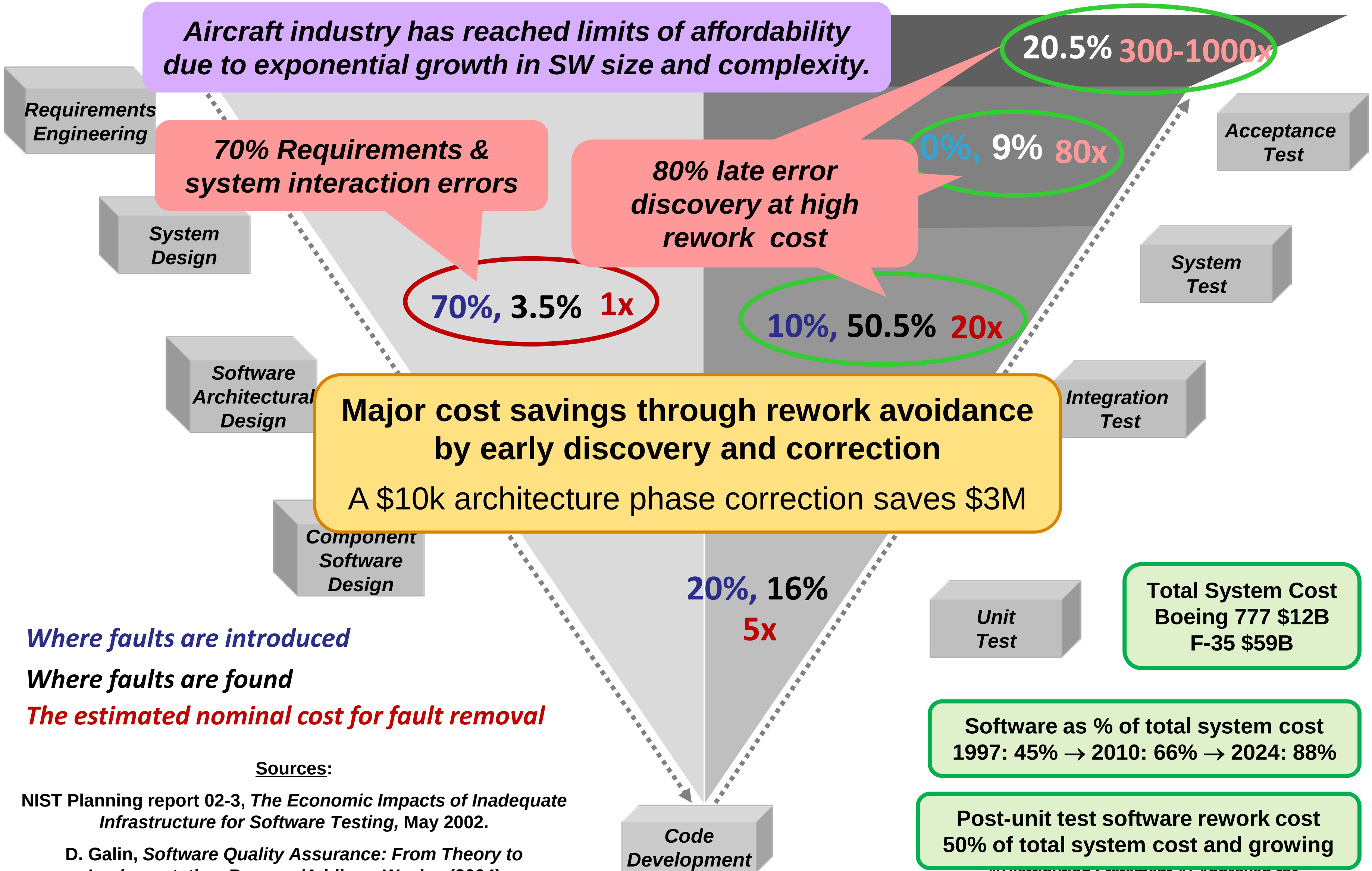
Lexus recently duplicated the problem on its own test track and developed a software upgrade for the vehicle's ESC system that would prevent the problem from happening. Dealers received the software fix last week and began notifying GX 460 owners to bring their vehicles in for repair.

We contacted the Lexus dealership from which we had anonymously bought the vehicle and made an appointment to have the recall work performed. The work took about an hour and a half.

Following that, we again put the SUV through our full series of emergency handling tests. This time, the ESC system intervened earlier and its rear did not slide out in the lift-off oversteer test. Instead, the vehicle understeered—or plowed—when it exceeded its limits of traction, which is a more common result and makes the vehicle more predictable and less likely to roll over. Overall, we did not experience any safety concerns with the corrected GX 460 in our handling tests.

How do you upgrade washing machine software?

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)

Current Industry Practice in DO-178B Compliant Requirements Capture

Industry Survey in 2009 FAA Requirements Engineering Study

Notation

Enter an "x" in every row/column cell that applies

Primarily textual "shall" requirement statements

	Sys…equirements	Dat…rconnect {ICD}	Hig…vel Software Requireme…	Low…el Software Requiremer…	Har…e Requirements
English Text or Shall Statements	39	27	36	32	29
Tables and Diagrams	31	30	30	19	18
UML Use Cases	1		2	4	
UML Sequence Diagrams			3	6	
UML State Diagrams			1	7	
Executable Models (e.g. Simulink, SCADE Suite, etc.)	7	1	8	8	1
Data Flow Diagrams (e.g. Yourdon)	4		6	9	
Other (Specify)-Proprietary Database, DOORS objects	1	4	2	2	1
Other (Specify)XML		1			
Operational models or prototypes	1	1			1
UML			1	1	

Tool

Enter an "x" in every row/column cell that applies	System Requirements	Data Interconnect {ICD}	High-Level Software Requirements	Low-Level Software Requirements	Hardware Requirements
Database (e.g., Microsoft Access)	3	4	3	3	
DOORS	23	13	22	18	12
Rational ROSE®			1	3	
RDD-100®					
Requisite Pro®	5	3	5	4	4
Rhapsody	1				
SCADE Suite	2		3	1	
Simulink	5	1	5	3	1
Slate	1		1	1	
Spreadsheet (e.g., Microsoft Excel)	5	4	5	4	3
Statemate					
Word Processor (e.g., Microsoft Word)	19	20	18	17	16
VAPS™		1	3	3	
Designer's Workbench™			1	1	
Proprietary Database, SCADE like pic tool		1	1		
Interleaf	1	1	1	1	1
BEACON	1	1	1	1	
CaliberRM	1	1	1	1	1
XM:		1			
Wiring diagram		1			1

Textual Requirement Quality Challenge

There is more to requirements quality than "shall"s and stakeholder traceability

IEEE 830-1998 Recommended Practice for SW Requirements Specification

Requirements error	%
Incomplete	21%
Missing	33%
Incorrect	24%
Ambiguous	6%
Inconsistent	5%

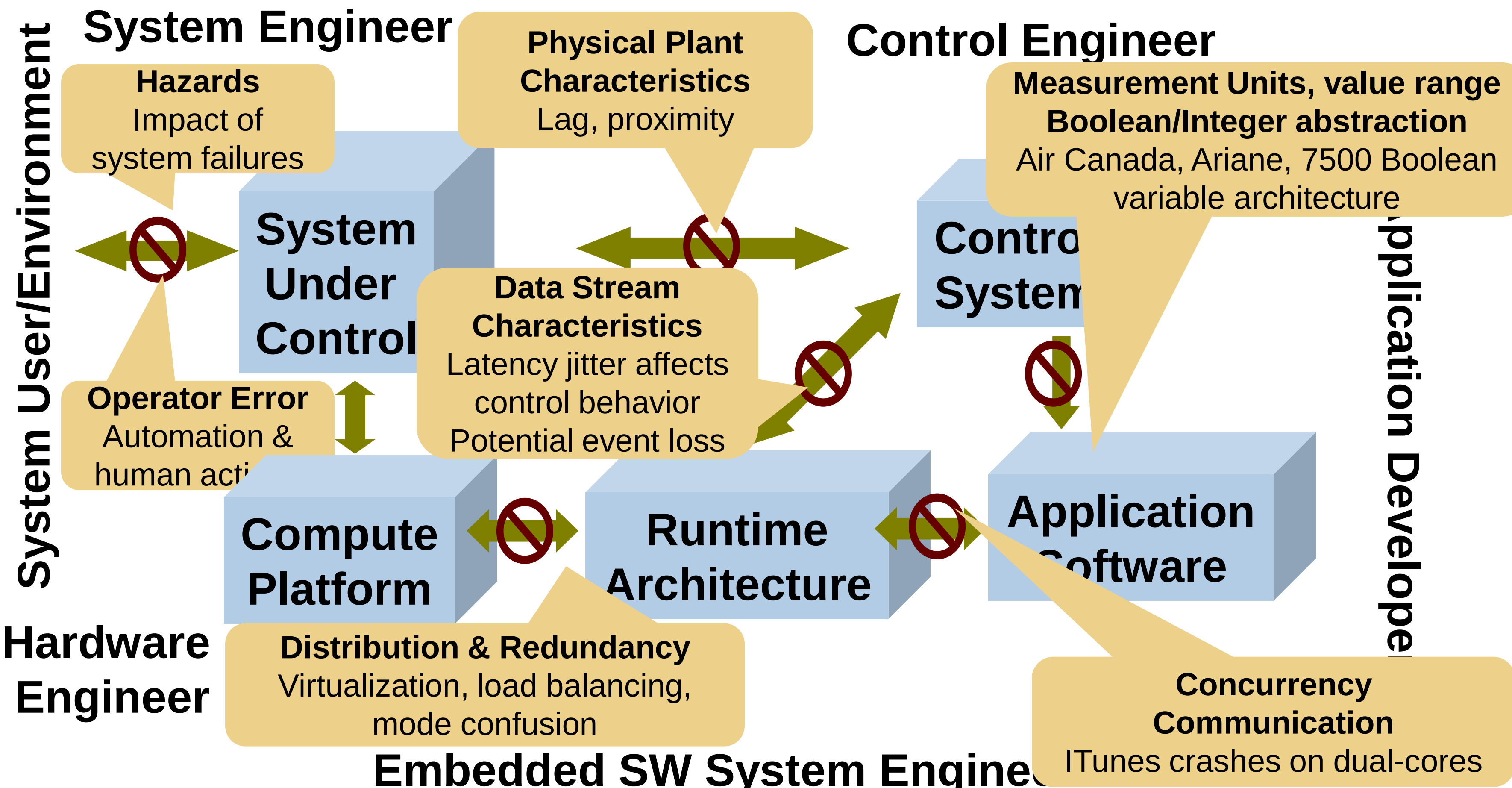
Browsable links/Coverage metrics

System to SW requirements gap [Boehm 2006]

How do we verify low level SW requirements against system requirements?

When StartUpComplete is TRUE in both FADECs and SlowStartupComplete is FALSE, the FADECStartupSW shall set SlowStartupInComplete to TRUE

Mismatched Assumptions in System Interactions



Embedded software system as major source of hazards

Why do system level failures still occur despite fault tolerance techniques being deployed in systems? Software system as hazard contributor

System Level Fault Root Causes

Violation of data stream assumptions

- Stream miss rates, Mismatched data representation, Latency jitter & age

Partitions as Isolation Regions

- Space, time, and bandwidth partitioning
- Isolation not guaranteed due to undocumented resource sharing
- fault containment, security levels, safety levels, distribution

Virtualization of time & resources

- Logical vs. physical redundancy
- Time stamping of data & asynchronous systems

Inconsistent System States & Interactions

- Modal systems with modal components
- Concurrency & redundancy management
- Application level interaction protocols

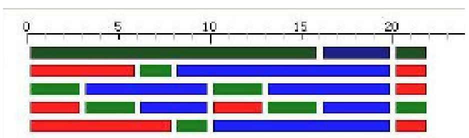
Performance impedance mismatches

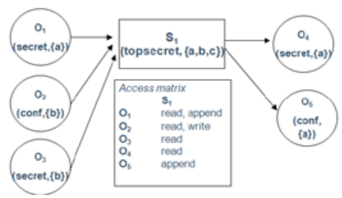
- Processor, memory & network resources
- Compositional & replacement performance mismatches
- Unmanaged computer system resources

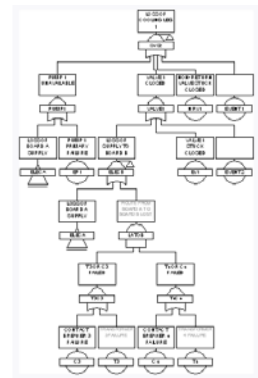
Model-based Engineering Pitfalls

The system

Inconsistency between independently developed analytical models

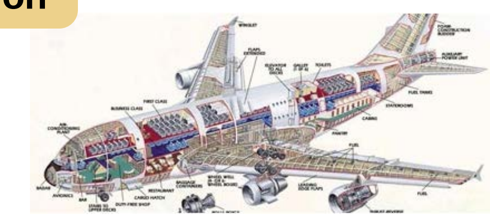






System models

Confidence that model reflects implementation



System implementation

This aircraft industry experience has led to the System Architecture Virtual Integration (SAVI) initiative

Outline: AADL Standard & ACVIP

- Challenges in embedded software systems
- Modeling-driven and architecture-centric engineering
- Overview of SAE AADL Standard suite
- AADL Language Overview
- AADL Tools
- Summary

What is Software Architecture?

The **software architecture** of a program or computing system is the structure or structures of the ***system*****,** which is:

- comprised of software components
- the externally visible properties of those components, and
- the relationships between them. [1]

A software system architecture consists of a set of

- communicating tasks,
- mapped onto a hardware platform, and
- interfacing with a physical target system or operational environment.

"externally visible properties" refers to those assumptions other components make of a component, such as a provided service, performance characteristic, fault handling, etc.

To allow for analysis, these 'externally visible properties' are precisely defined in the AADL.

Architecture serves as the basis for system analysis.

[1] Documenting Software Architectures, Addison Wesley, 2010

Why UML, SysML Are Not Sufficient

- System engineering
  - Focus on system architecture and operational environment
  - SysML developed to capture interactions with outside world, as a standardized UML profile
  - 4 pillars/diagrams: requirements, parameterics (added in SysML), structure, behavior
- Conceptual architecture
  - UML-based component model
  - Architecture views (DoDAF, IEEE 1471)
  - Platform Independent model (PIM)
- Embedded software system engineering
  - SAE AADL with well-defined semantics for SW, runtime, computer, physical system architectures
  - OMG Modeling and Analysis of Real Time Embedded systems (MARTE) as UML profile leveraging AADL semantic Meta model
  - Multiple analysis perspectives in Model-Based Engineering
  - xUML insufficient for PSM (Kennedy-Carter, NATO ALWI study)

What is the AADL?

SAE International Architecture Analysis and Design Language (AADL) is

an industry standard* notation

for modeling embedded software system architectures

That supports architectural analysis of functional and operational quality attributes, virtual system integration, and construction from verified models

for the avionics, aerospace, automotive, and medical device domains.

AADL

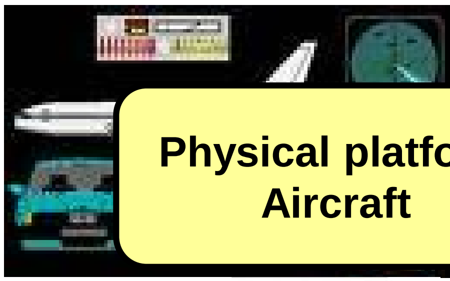
- Is based on 15 Years of DARPA funded ***research*** technologies
- Was first published Nov 2004 and revised in Jan 2009 *(**V2**) and Sept 2012 (**V2.1**)*

SAE International

**** SAE International standard document AS 5506B (R)***

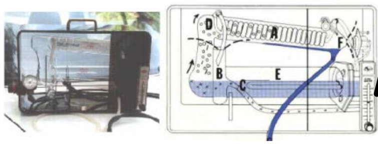
SAE Architecture Analysis & Design Language (AADL) for Embedded Systems

The System



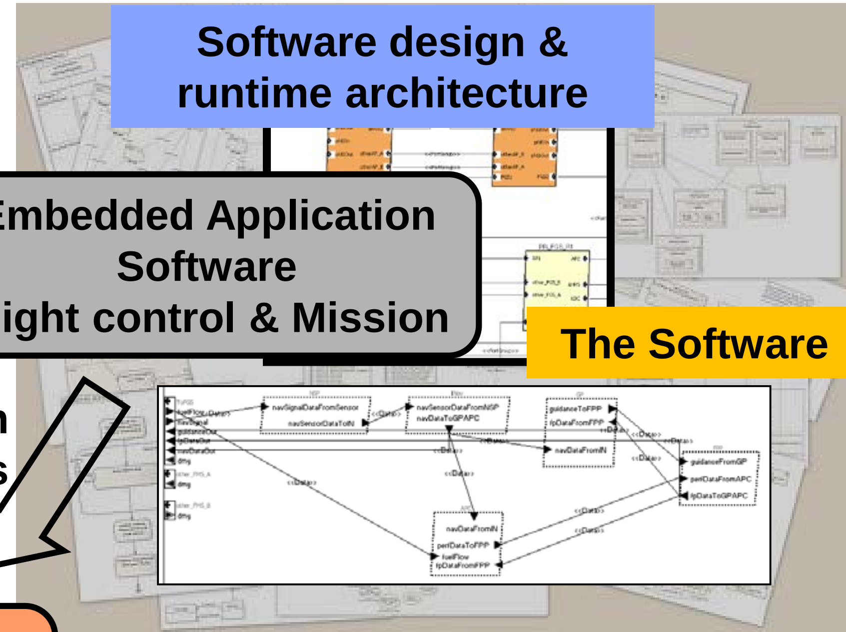
Physical platform
Aircraft

Continuous Distiller



Control
Guidance

Software design & runtime architecture



Embedded Application Software
Flight control & Mission

The Software

Deployed on
Utilizes

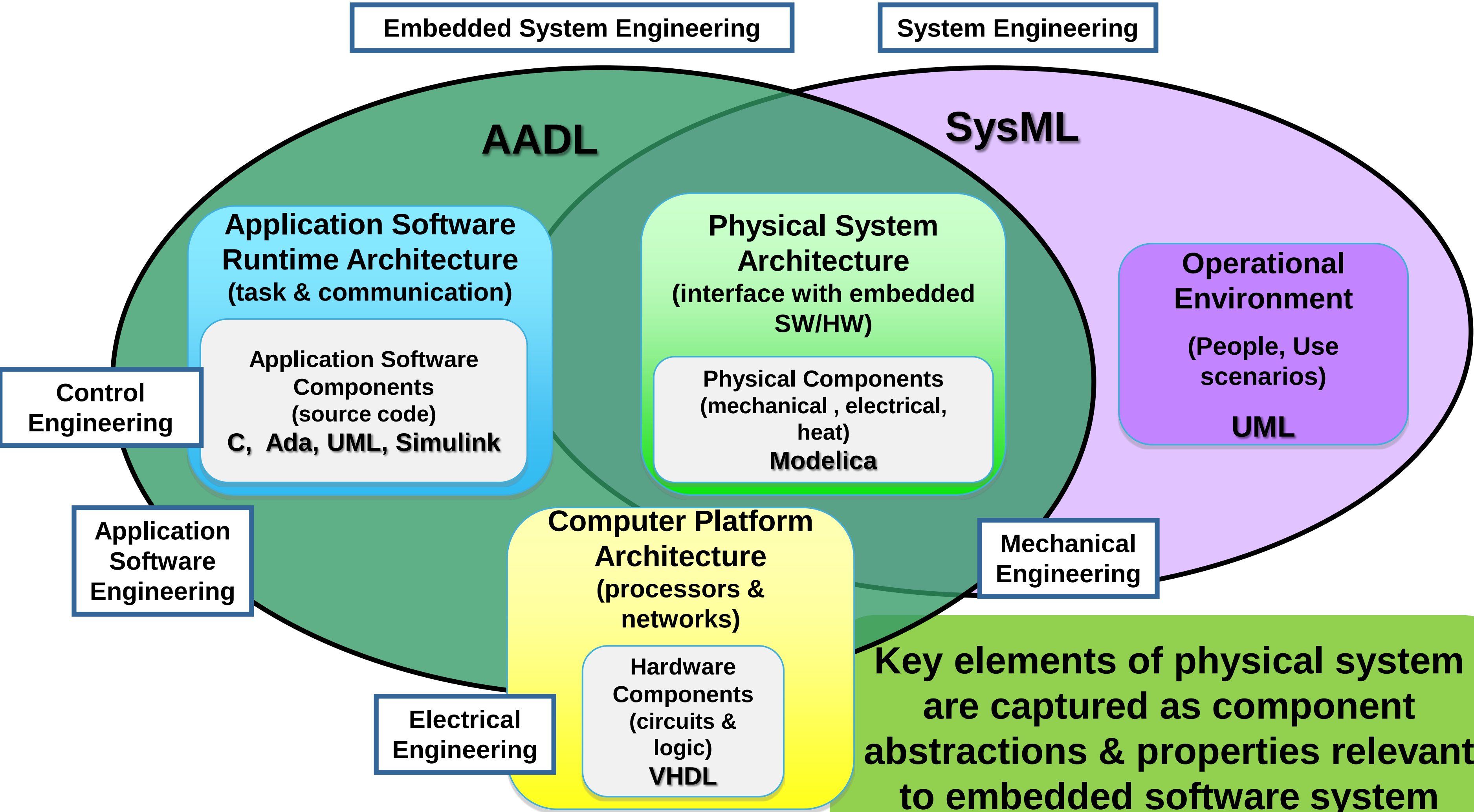
Physical interface
Platform component

Computer System
Hardware & OS

The Computer System

AADL focuses on interaction between the three major elements of a software-intensive system based on architectural abstractions of each.

Cooperative Engineering of Systems

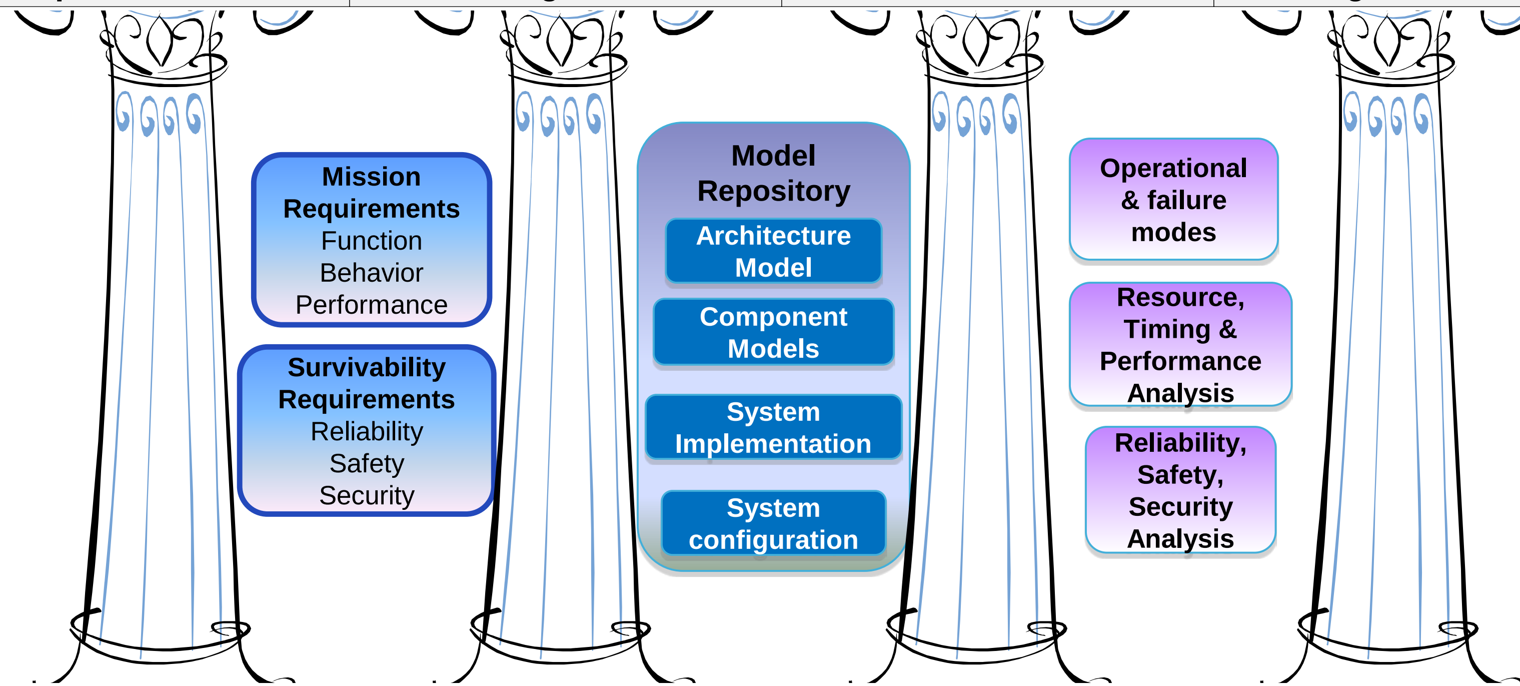


Key elements of physical system are captured as component abstractions & properties relevant to embedded software system analysis

Reliability & Qualification Improvement Strategy

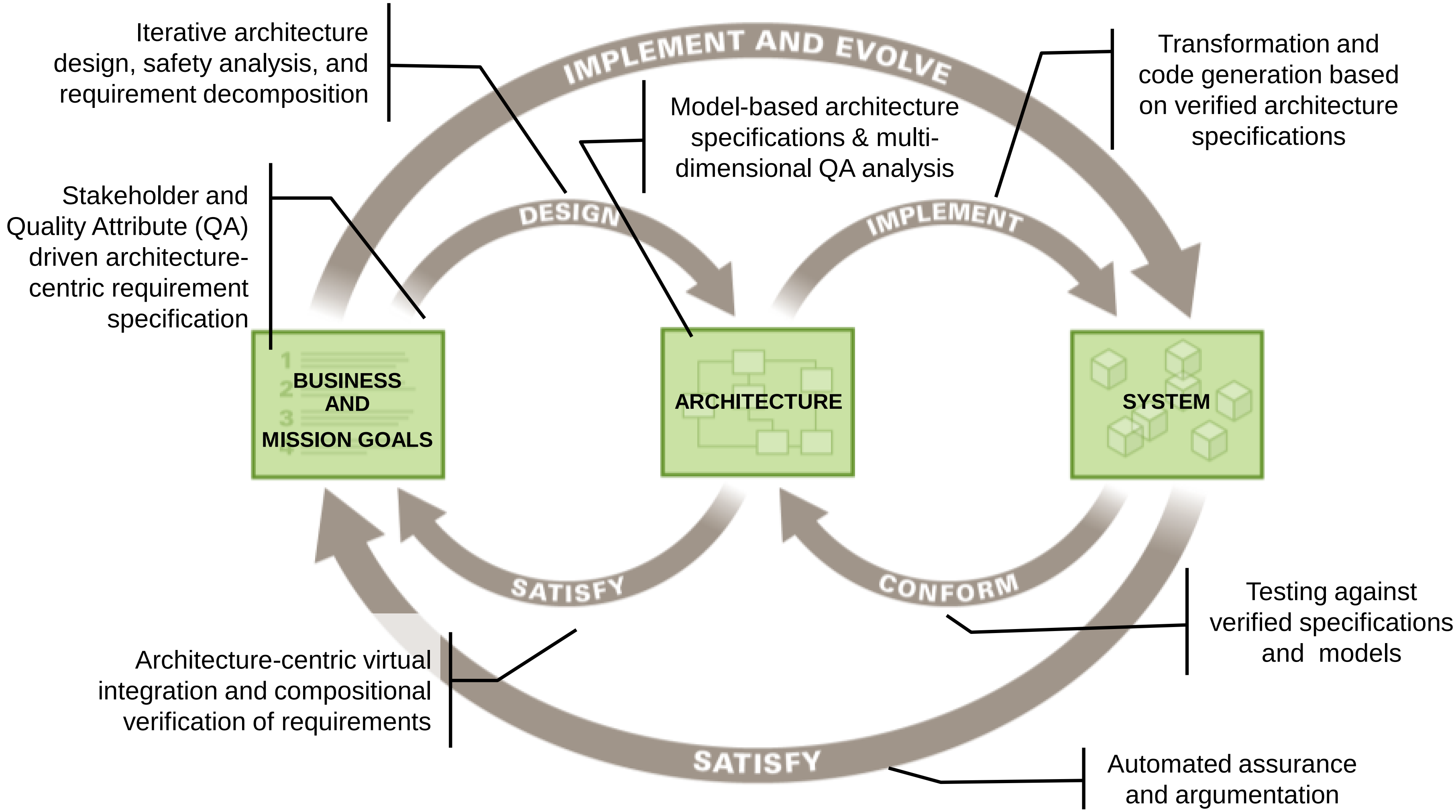
2010 SEI Study for AMRDEC
Aviation Engineering Directorate

Architecture-led Requirement Specification	Architecture-centric Virtual System Integration	Static Analysis & Compositional Verification	Incremental Assurance Plans & Cases throughout Life Cycle

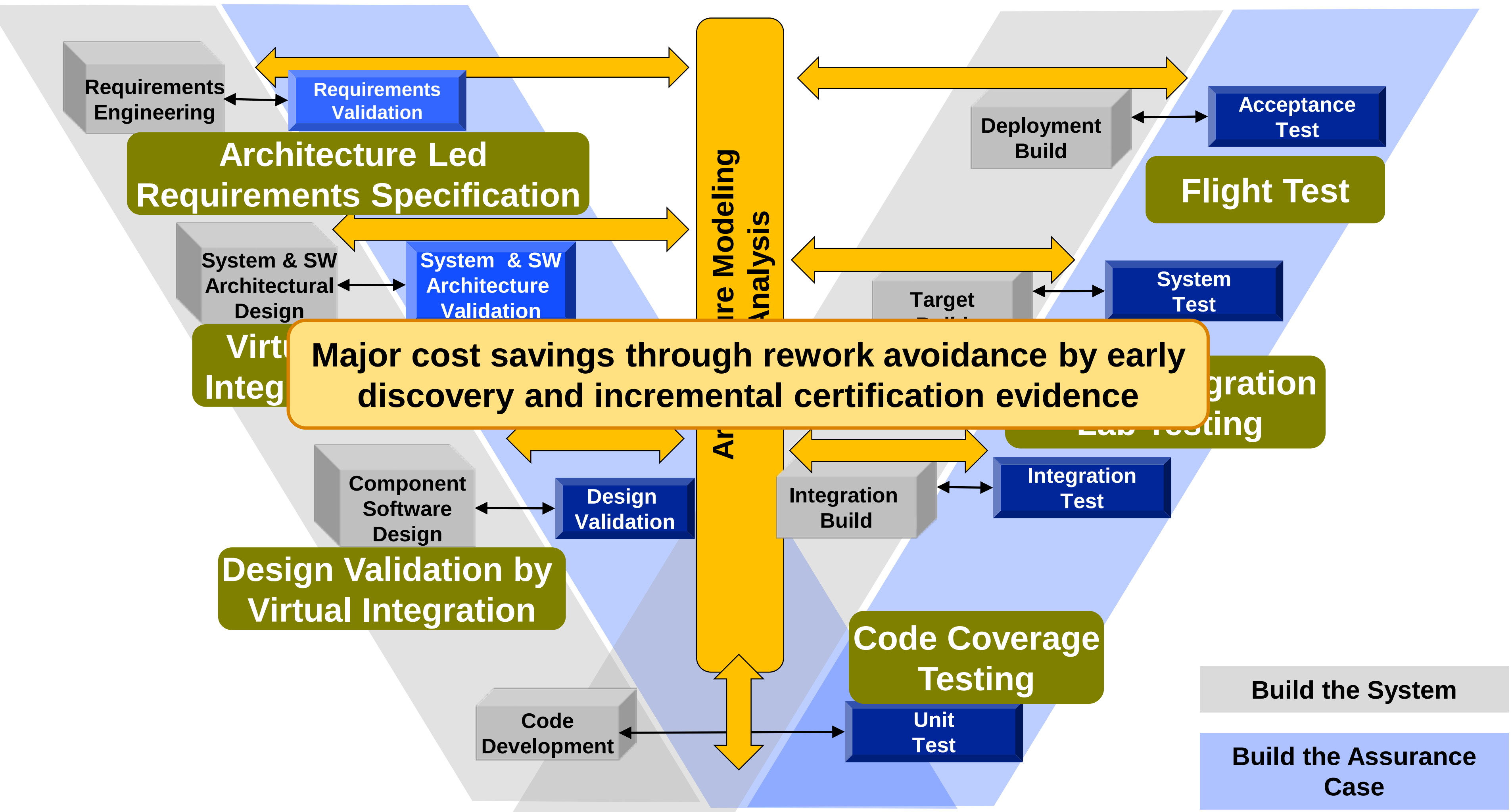


Four pillars for Improving Quality of Critical Software-reliant Systems

Architecture-centric Virtual Integration Practice (ACVIP)



Building the Assurance Case throughout the Life Cycle



Outline: AADL Standard & ACVIP

- Challenges in embedded software systems
- Modeling-driven and architecture-centric engineering
- Overview of SAE AADL Standard suite
- AADL Language Overview
- AADL Tools
- Summary

The SAE AADL Standard Suite (AS-5506 series)

Core AADL language standard (V2.1-Sep 2012, V1-Nov 2004)

- Strongly typed language with well-defined semantics
- Textual and graphical notation
- Standardized XMI interchange format

Standardized AADL Extensions

Error Model language for safety, reliability, security analysis
ARINC653 extension for partitioned architectures
Behavior Specification Language for modes and interaction behavior
Data Modeling extension for interfacing with data models (UML, ASN.1, …)

AADL Extensions in Progress

Requirements Definition and Assurance Language
Synchronous System Specification Language
Hybrid System Specification Language
System Constraint Specification Language

AADL: The Language

Precise execution semantics for components

- Thread, process, data, subprogram, system, processor, memory, bus, device, virtual processor, virtual bus

Continuous control & event response processing

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

Operational modes & fault tolerant configurations

- Modes & mode transition

Modeling of large-scale systems

- Component variants, layered system modeling, packaging, abstract, prototype, parameterized templates, arrays of components, connection patterns

Accommodation of diverse analysis needs

- Extension mechanism, standardized extensions

System Level Fault Root Causes

Violation of data stream assumptions

- Stream miss rates, Mismatched data representation, Latency jitter & age

End-to-end latency analysis
Port connection consistency

Partitions as Isolation Regions

- Space, time, and bandwidth partitioning
- Isolation not guaranteed due to undocumented resource sharing
- fault containment, security levels, safety levels, distribution

Process and virtual processor to model partitioned architectures

Virtualization of time & resources

- Logical vs. physical redundancy
- Time stamping of data & asynchronous systems

Virtual processors & buses
Multiple time domains

Inconsistent System States & Interactions

- Modal systems with modal components
- Concurrency & redundancy management
- Application level interaction protocols

Operational and failure modes
Interaction behavior specification
Dynamic reconfiguration
Fault detection, isolation, recovery

Performance impedance mismatches

- Processor, memory & network resources
- Compositional & replacement performance mismatches
- Unmanaged computer system resources

Resource allocation & deployment configurations
Resource budget analysis & scheduling analysis

Codified in Virtual Upgrade Validation method

Architecture Views and SAE AADL

Component View

- Model of system composition & hierarchy
- Software, execution platform, and physical components
- Well-defined component interfaces

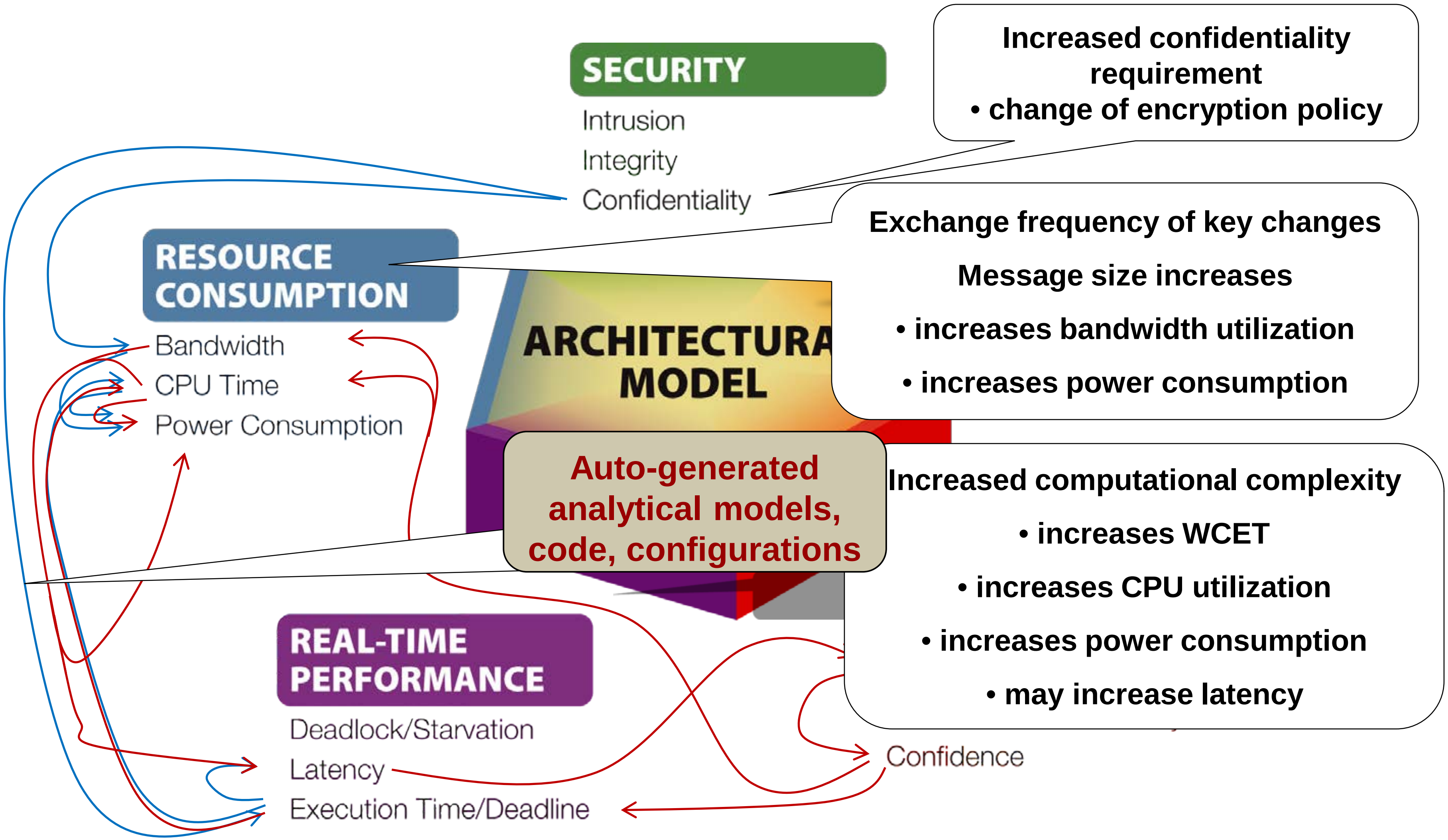
Concurrency & Interaction View

- Time ordering of data, messages, and events
- Dynamic operational behavior
- Explicit interaction paths & protocols

Deployment view

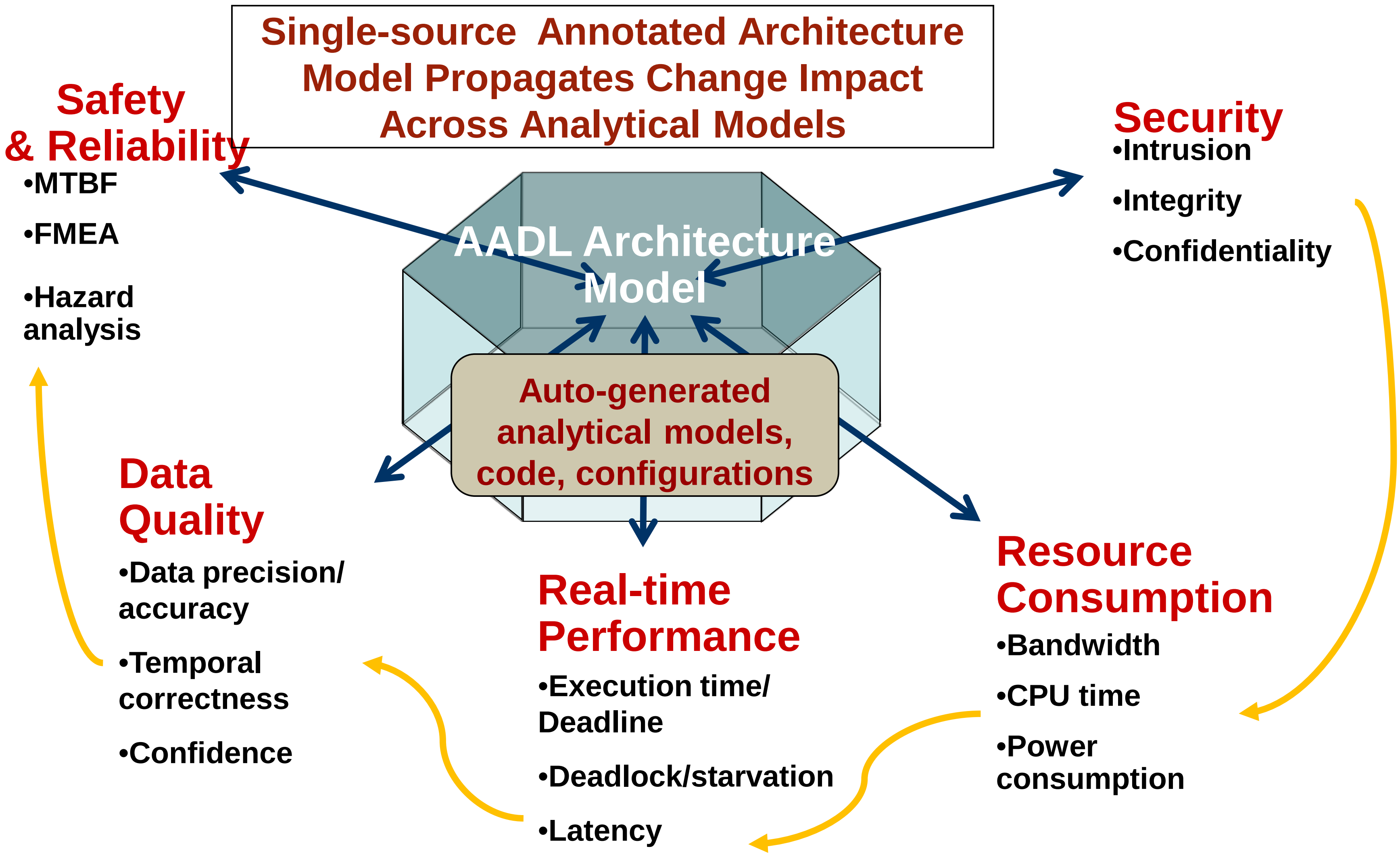
- Execution platform as resources
- Binding of application software
- Specification & analysis of runtime properties, …

Change Impact Across Analysis Dimensions

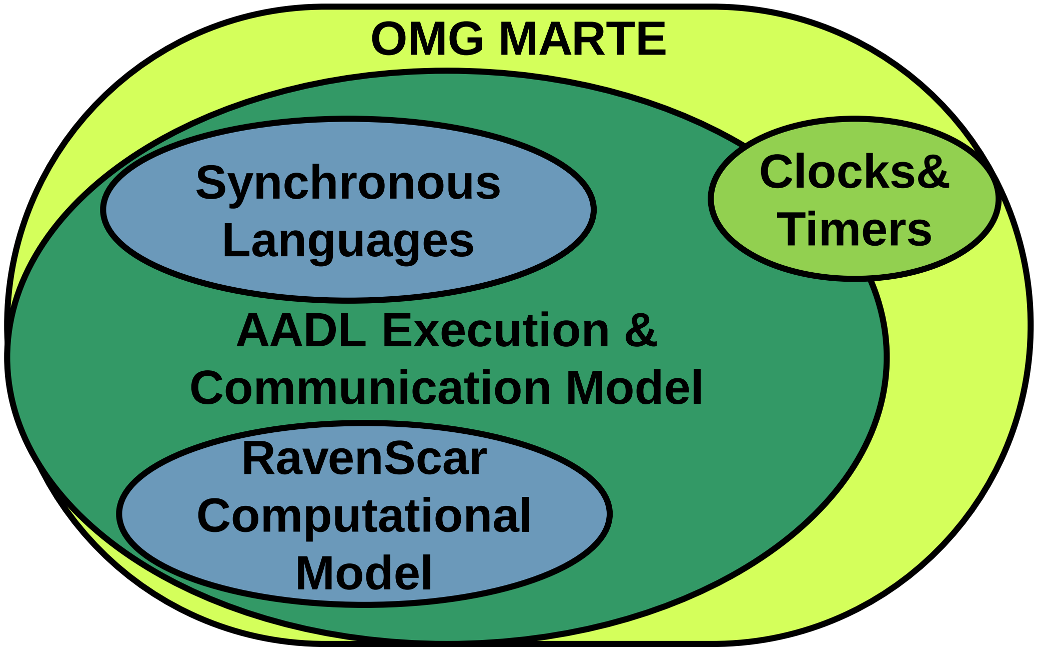


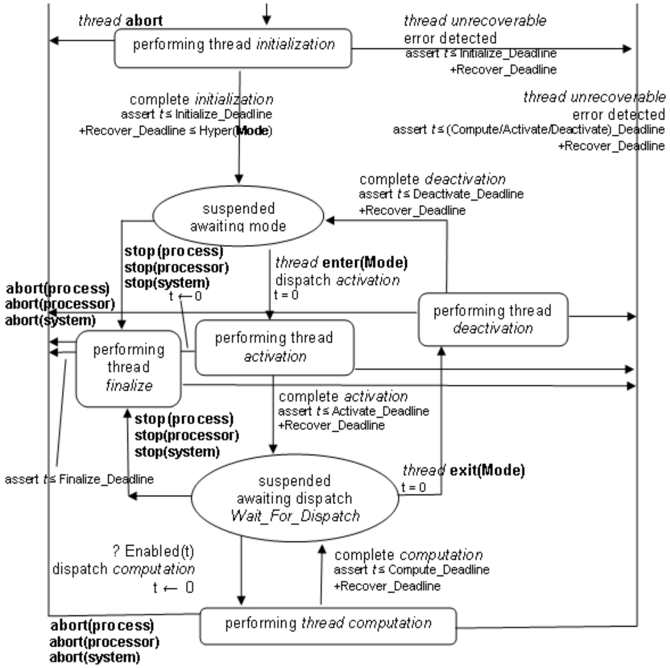
Single-Model, Multi-Dimensional Analysis

Change Impact Across Analysis Dimensions



Well-defined Execution Semantics





OMG MARTE

Focus on implementation

- Timers to trigger task execution
- Send/receive operations
- Behavioral states and transitions

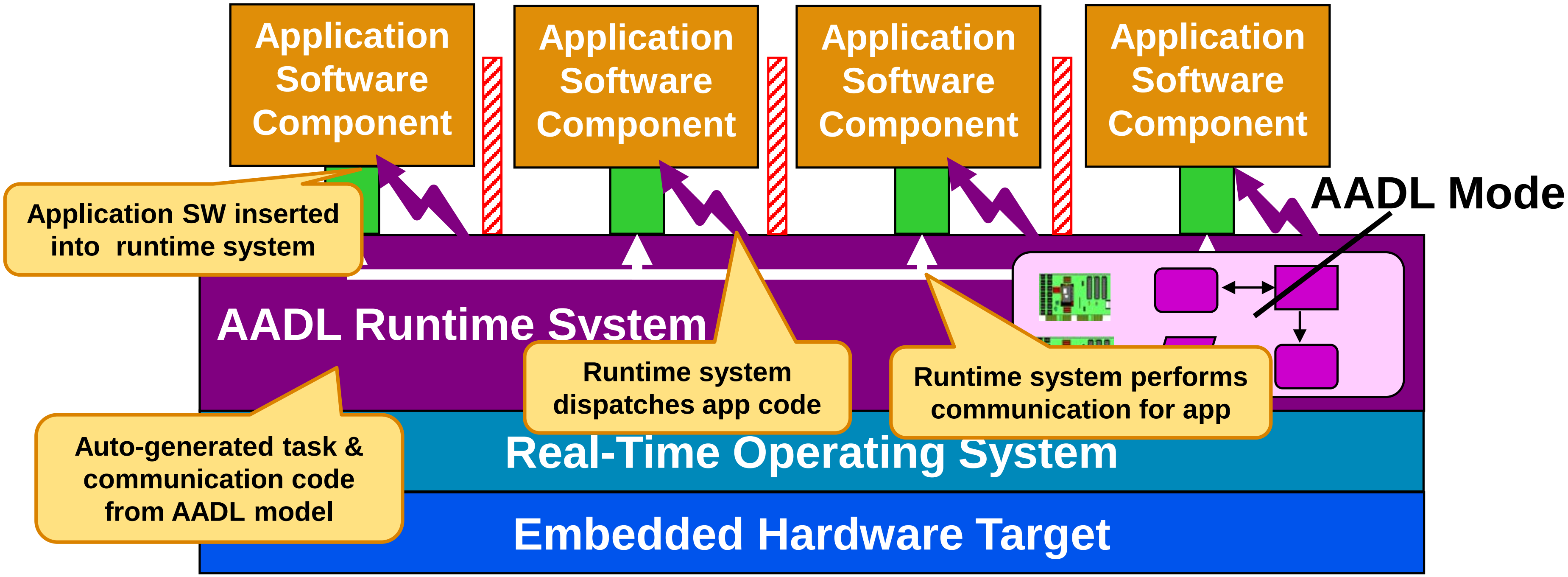
SAE AADL

Focus on Architecture Abstraction

- Thread execution
- Communication timing
- Operational modes & architecture reconfiguration

Partitioned Run-Time Architecture

A successful embedded systems is a layered runtime architecture that supports partitioning



Runtime exec is generated against a common RTOS and communication API

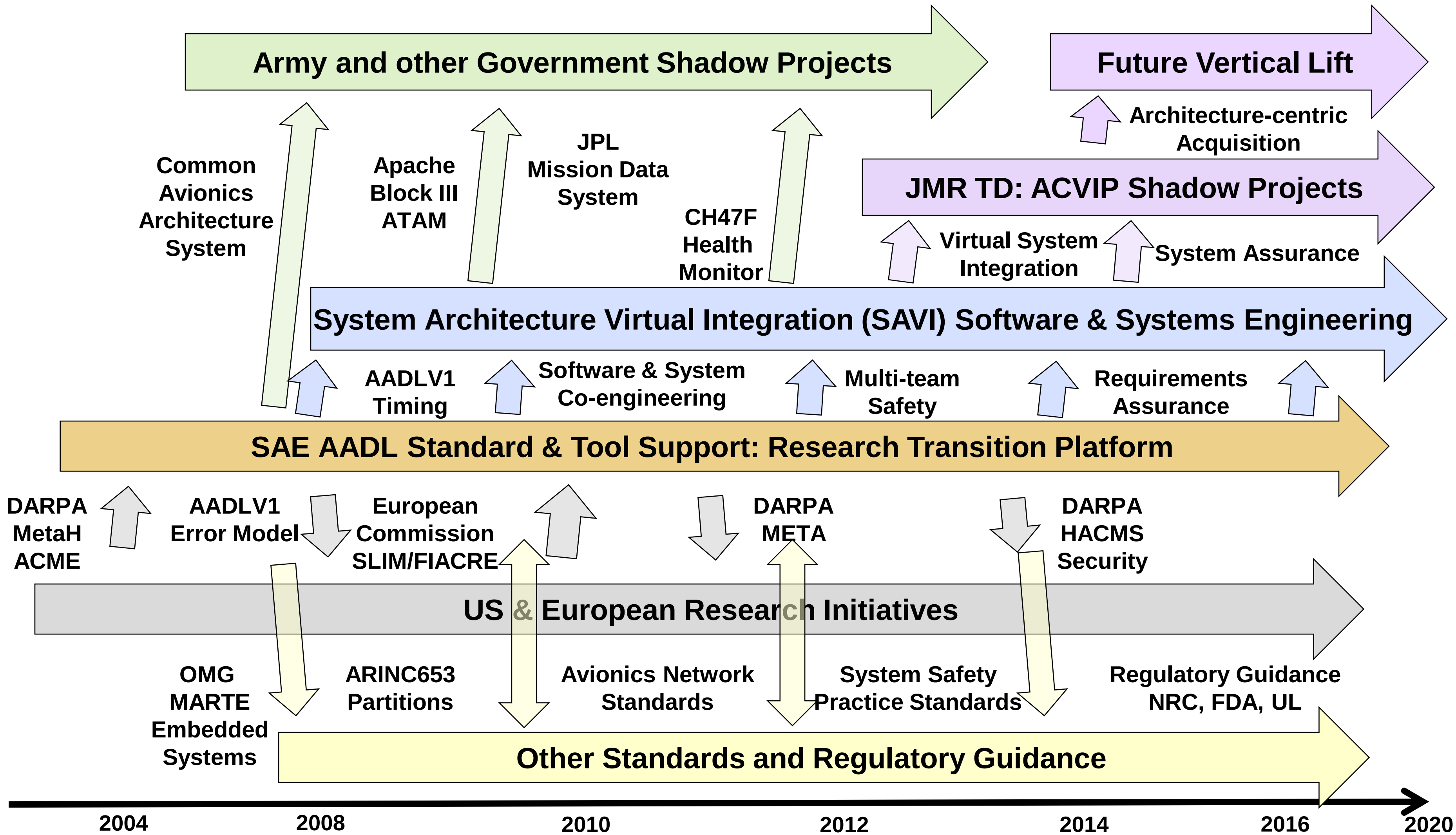
Strong Partitioning

- Timing Protection
- OS Call Restrictions
- Memory Protection

Interoperability/Portability

- Tailored Runtime Executive
- Standard RTOS API
- Application Components

AADL-based Virtual System Integration Technology Approach



Evolution, Maturation and Transition

International R&D Programs Leveraging SAE AADL

Compositional
Timing Framework
OSD, 500K, 2014

P Project
Auto Code Gen
2011-2014

OpenGroup
Real-Time Forum
EU + US partners
2008-current

D-MILS
Design of Secure Systems
2013 - $4.9M

OPEES
Formal analysis
2011-2014

ESA COMPASS
System SW Co-engineering
2008-current

TOPCASED
Open Source Embedded
Systems Tool Framework
28 partners €20+M 2005-2009

AVSI SAVI
Analysis-based System Validation
12 partners $20M 2008-current

IST ARTIST2
Embedded Systems
Center of Excellence
2007-2012

ITEA SPICES
Model-Driven Embedded
Systems Engineering
15 partners €16M 2006-2009

DARPA META, HACMS
Complex System
Engineering, Security
2010-current

Flex-eWare
Auto Code Generation
2007-2010

EC ASSERT
Proof-based Satellite
Architectures
ESA + 30 partners
€15M 2004-2007

ESA TASTE
System & SW
Validation & Generation
2010-current

PARSEC
Safety/security f
2010-2013

Integrated Clinical
Environment
Device Certification
FDA KSU
2011-current

MASIW
Avionics Workbench
2011-current
$2M per year

NASA CVFCS
AF SwPI
RC formal methods

PROARTIS
Partitioned RT systems
2010-2013 € 1.8M

RAMSES
Auto Code Generation
2012-current

Benefits of Architecture-centric Engineering

Reduce risks

- Analyze system early and throughout life cycle
- Understand system wide impact
- Validate assumptions across system

Increase confidence

- Validate models to complement integration testing
- Validate model assumptions in operational system
- Evolve system models in increasing fidelity

Reduce cost

- Fewer system integration problems
- Fewer validation steps through use of validated generators

Transition to Architecture Centric Virtual Integration

Build on architecture tradeoff analysis (e.g., SEI ATAM)

- Provides focused evaluation method
- MBE/AADL provides quantitative analysis & starter models to build on

Project reviews & root cause analysis

- Identify systemic risks in problem systems & in technology migration
- AADL provides semantic framework to identify issues and potential mitigation strategies

Architecture documentation of existing systems

- Leverage existing design data bases
- Challenge: abstract away from design details (“what” instead of “how”)

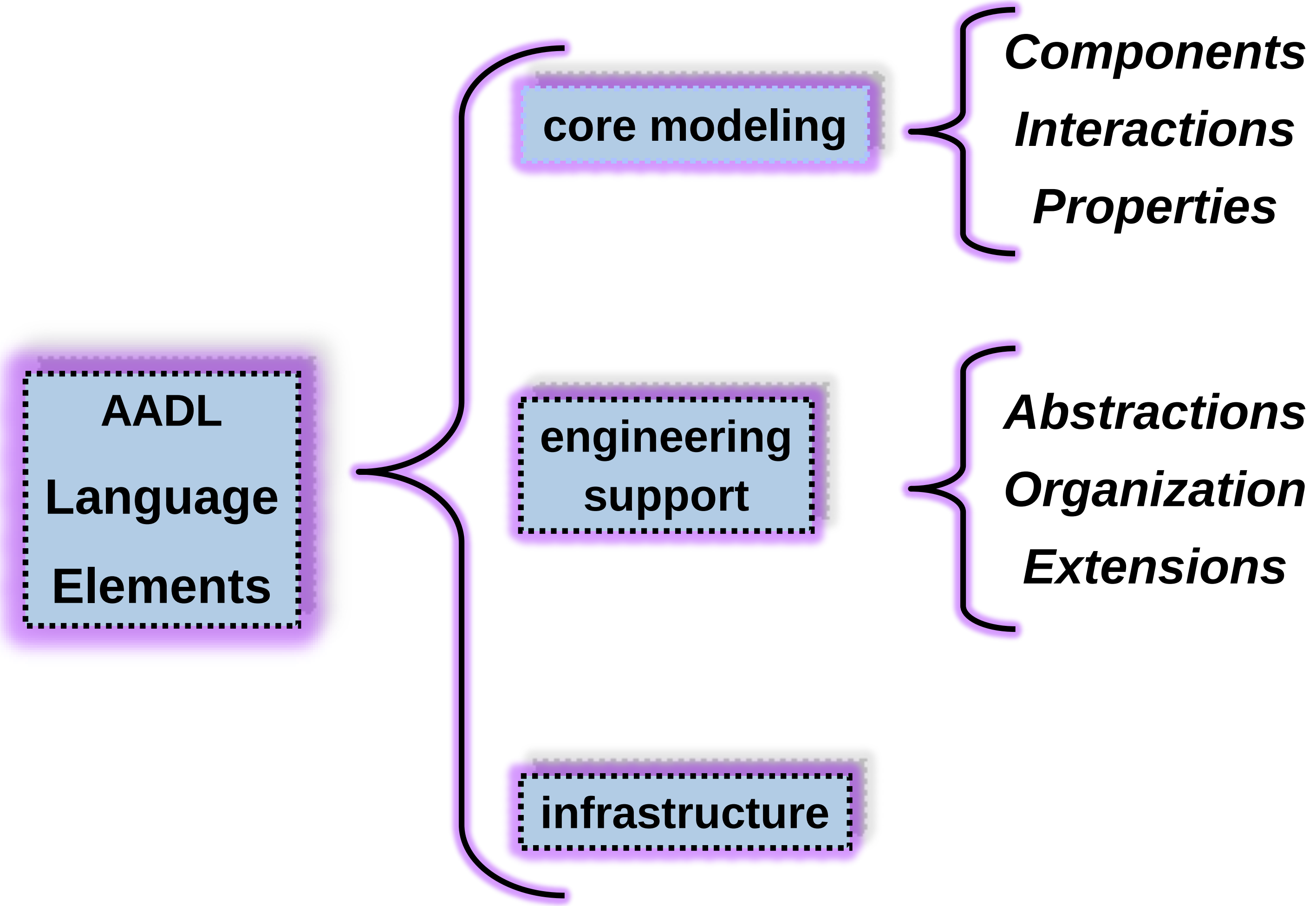
System and software assurance

- Provides structured approach to safety/dependability assurance
- MBE/AADL provides evidence based on validated models

Outline: AADL Standard & ACVIP

- Challenges in embedded software systems
- Modeling-driven and architecture-centric engineering
- Overview of SAE AADL Standard suite
- AADL Language Overview
- AADL Tools
- Summary

AADL Language Elements



Component-Based Representation

Specifies a well-formed interface

Component type allows for multiple implementations with extensions

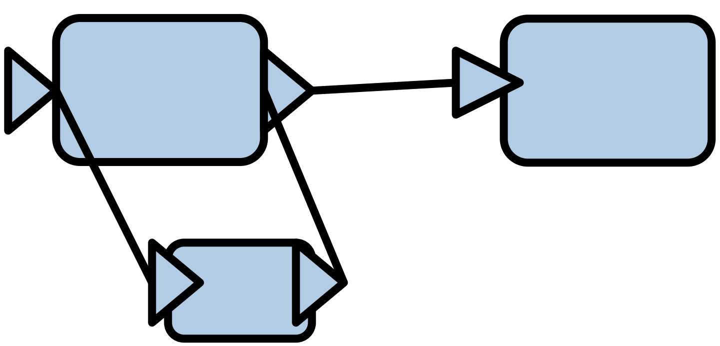
All external interaction points defined as ***features***

Data and event ***flows*** through component, across multiple components

Properties to specify component characteristics

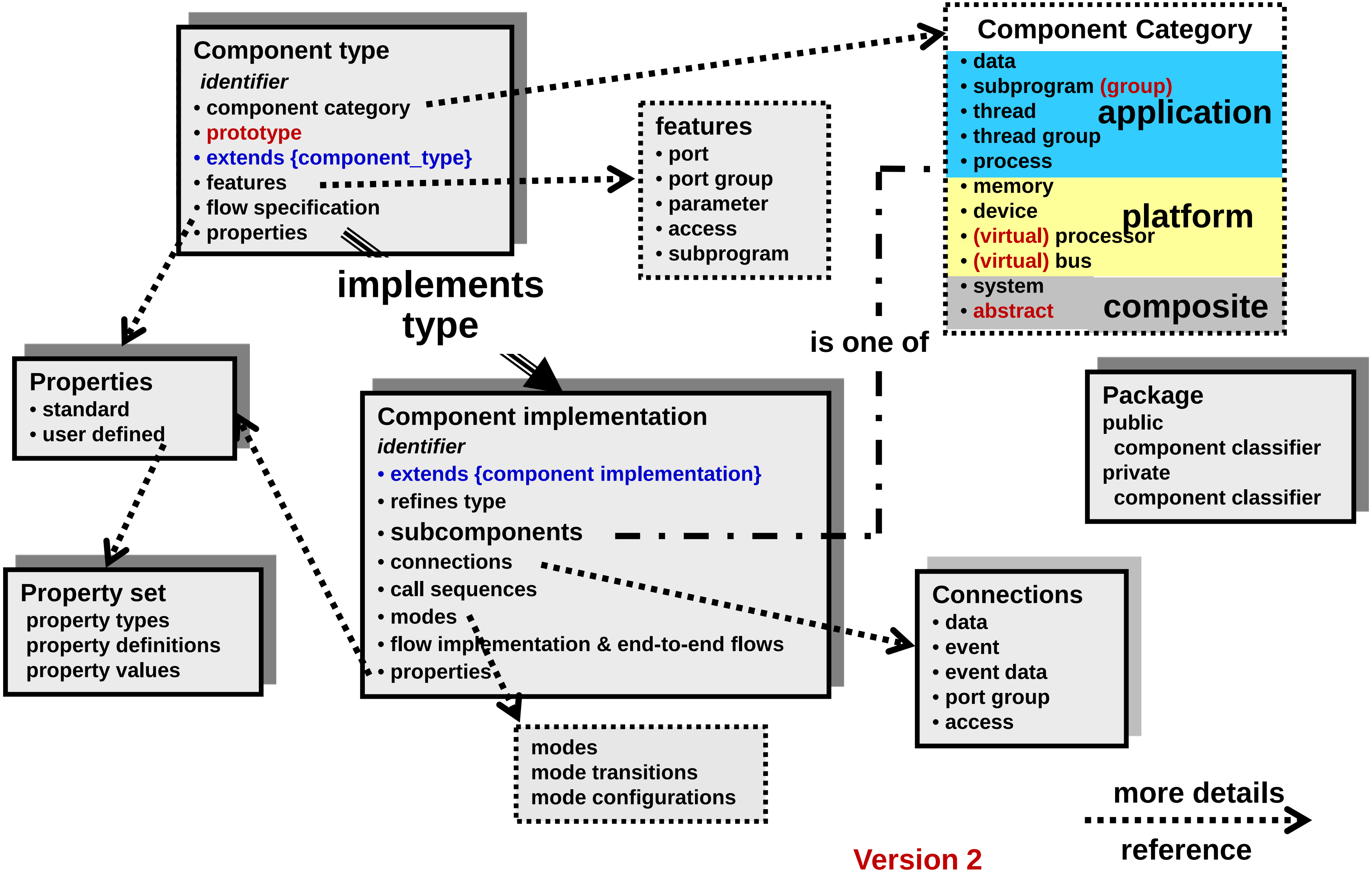
Components organized into system hierarchy

Component interaction declarations must follow system hierarchy



```
System my_system
     Features
     Flows
     Properties
End my_system;
System implementation my_system2
End my_system2;
```

AADL: Components and Connections



Application Software Components

System – hierarchical organization of components

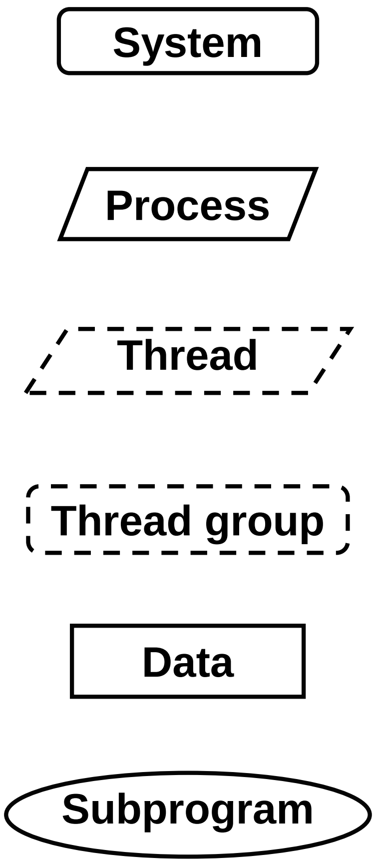
Process – protected address space

Thread – a schedulable unit of concurrent execution

Thread group – logical organization of threads

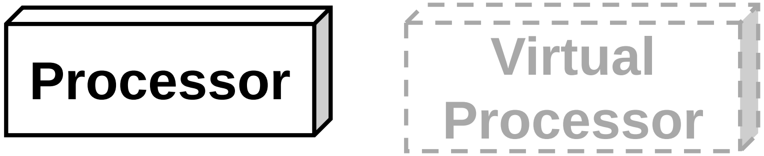
Data – potentially sharable data

Subprogram – callable unit of sequential code

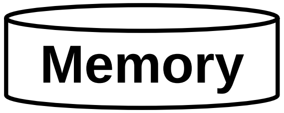


Execution Platform Components and Devices

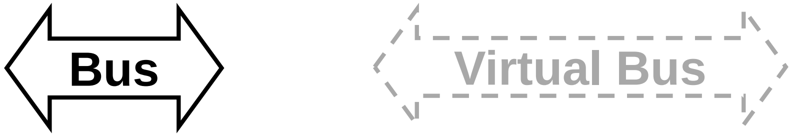
Processor / Virtual Processor – Provides thread scheduling and execution services



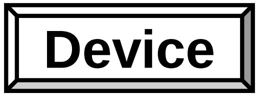
Memory – provides storage for data and source code



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



Device – interface to external environment



AADL Language Concepts 1

Component – an entity representing an abstraction of hardware, software, or a system.

Type – A declaration that specifies the functional interfaces of a component.

- All components must have a type declaration
- Types allow the specification of component for syntax checking

A ‘type’ can be thought of as a template for a modeled component

Types declarations may be empty or incomplete

One component type may extend another component type

Typical uses of component types

- Generic specification of a modeling component (an empty type)
- Base representation for components with optional/incomplete features, e.g. a family of components with a common set of interfaces.

```
system engine_monitor
  features
    engine_RPM: in data port;
    engine_overspeed: out data port;
end engine_monitor;
```

AADL Language Concepts 2

Implementation – Is the realization of the associated component type. It is compliant with its corresponding type declared interfaces.

- Indentified by the reserved word ‘implementation’

A ‘implementation’ can be though of as the realization of the component type

Implementation may be empty e.g. directly implement the type

There may be many implementations based on various subsets of component types, the connections among them, and various properties of the implementation.

Uses of component implementations

- Directly implement a component type
- Represent an analysis model based on the composition of component types.

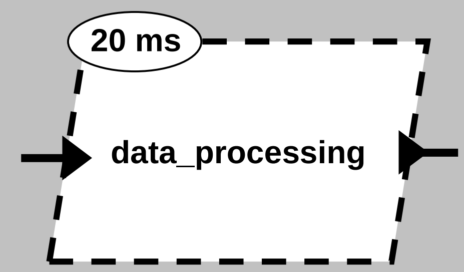
```
system implementation engine_monitor.impl
-- a simple implementation
end engine_monitor.impl;
```

AADL Representation Forms

AADL Text

```
thread data_processing
features
  raw_speed_in: in data port;
  speed_out: out data port;
properties
  Period => 20 ms;
end data_processing;
```

Graphical



XML

```
<ownedThreadType name="data_processing">
  <ownedDataPort name="raw_speed_in"/>
  <ownedDataPort name="speed_out" direction="out"/>
  <ownedPropertyAssociation property="Period"
    <ownedValue xsi:type="aadl2:IntegerLiteral"
        value="20" unit="ms"
    </ownedValue>
  </ownedPropertyAssociation>
</ownedThreadType>
```

Outline: AADL Standard & ACVIP

- Challenges in embedded software systems
- Modeling-driven and architecture-centric engineering
- Overview of SAE AADL Standard suite
- AADL Language Overview
- ➡ AADL Tools
- Summary

AADL Tool Support

Open Source AADL Tool Environment (OSATE) by SEI

- Eclipse-based IDE for AADL and Annexes, Multiple analysis plugins
- Reference implementation for core AADL and annexes
- Vehicle for in-house prototyping and for architecture research

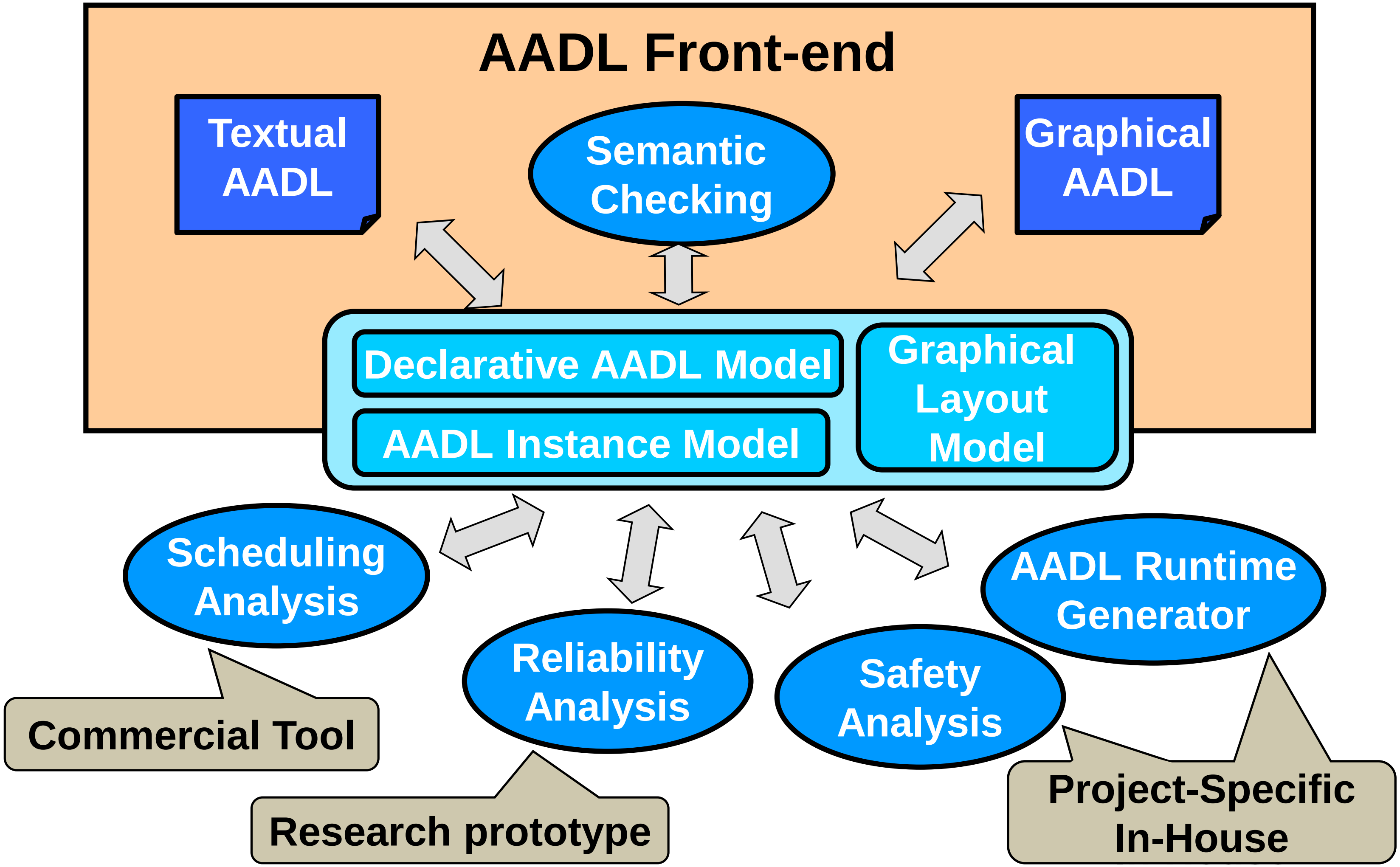
EllisDiss

- STOOD for AADL (http://www.ellidiss.com/products/stood/)
  - A development environment/tool chain that is supported by AADL, UML 2.0, HRT-HOOD, Requirements Analysis, and Software Method Prototyping
  - Features support for requirements capture/traceability, architectural design, and detailed design.
- AADL Inspector (http://www.ellidiss.com/products/aadl-inspector/)
  - Applies static syntax & legality rule checker, schedualibility analysis – turnkey integration to current version of CHEDDAR

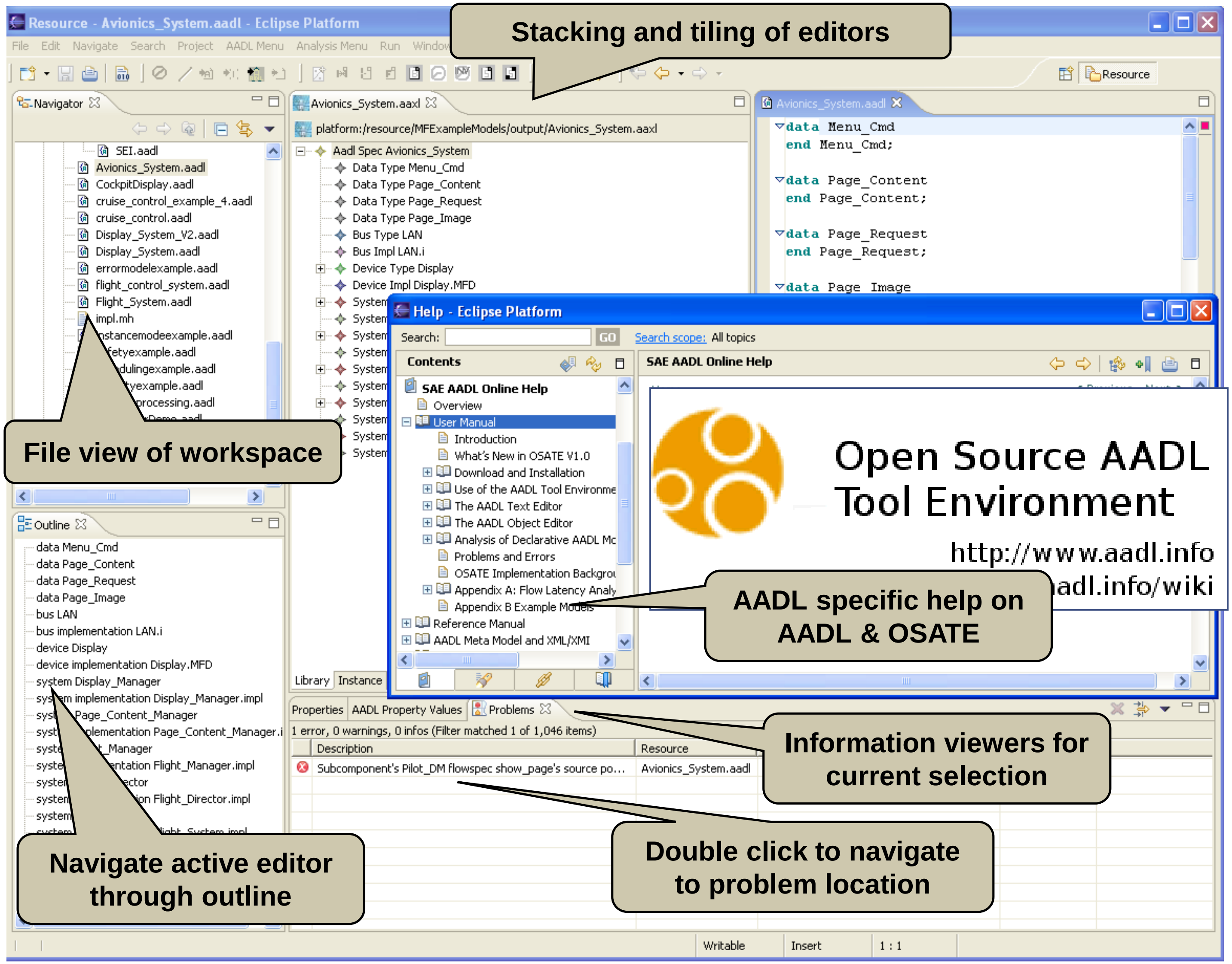
MASIW (ISPRAS)

- https://forge.ispras.ru/projects/masiw-oss
- an open source Eclipse-based IDE for development and analysis of AADL models

XML-Based Tool Integration Strategy



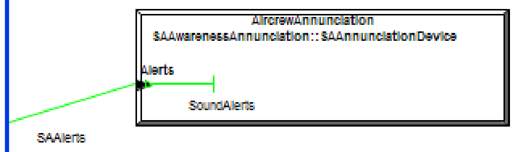
Open Source AADL Tool Environment - OSATE



AADL-based Requirement Specification

```
system ASSASystem
features
    AMPSInterface : in out event data port MissionSystemDataTypes::PlanningInformation;
    IncomingCOP : in data port;
    ThreatAlerts : out data port;
    WireObstaclesForAvoidance : out data port TrackTypes::ObstacleTrackSet {
        Data_Model::Base_Type => (classifier (TrackTypes::ObstacleTrack));
    };
    GeospatialData : in data port;
    EnvironmentalInformation : in data port;
    Weather: in data port;
    OwnAircraftPosition: in data port MissionSystemDataTypes::Position;
    SSSAirCrewPresentation : out feature group SAAwarenessAnnunciation::AirCrewSAInformation;
end ASSASystem;
```

AADL Model Acts as Requirement Specification




```
data WWTrack
properties
    acvip::aliases => ("Hostile Fire Detection message");
    JMRMIS::ObservedObjects => ( classifier(SAObservations::BallisticWeapons
end WWTrack;

data WW
propert
    Dat
    Dat
end WWT
```

ACVIP and Project Specific Properties

ACVIP
1 OutputInterval => 100 ms
Communication_Properties
Data_Model
(...) Base_Type => (classifier (WWTrack))
(...) Dimension => (JMRMISConstants::MaxWWTracks)

Display
Information

P
liases => ("ASE", "AST system", "AS system")
_OP_Properties
MIS
ensorKind => Passive
bservedObjects => (classifier(SAObservations::BallisticWeaponsFire))
bservationRadius => NM
ng_Properties
eadline => Period
1 Period => ms

Basis for assurance plans and assurance cases

Problems | Assurance Case

	Verified	Level (%)	Risk
Requirements Group SafetyHazards		NaN	
Requirements Group ACT		NaN	
Requirement ACT-SB-REQ2: queue size zero and abort overflow		100.0	!
Requirement ACT-SS_REQ9: Homing command results in SMM		NaN	!
Requirement ACT-OG-REQ5: MaxStepCount of 15 is used as step		100.0	!
Requirement ACT-SB-REQ6: StepCount == zero when reset to n		100.0	!
Requirement ACT-IA-REQ7: Stepcount within range		NaN	!
Requirement ACT-SS-REQ1: command arrival driven command		100.0	!
Requirement ACT-SB-REQ4: StepCount == # of step signals to n		NaN	!
Requirement ACT-IA-REQ8: Steprate is MaxStepCount		NaN	!
Requirement ACT-IC-REQ3: data representation for commandin		100.0	!
Requirements Group Motor		NaN	

Open Source AADL Analysis Tools - 1

ASSERT/TASTE: European Space Agency, tools dedicated to the development of embedded, real-time systems

http://taste.tuxfamily.org/wiki/index.php?title=Overview

COMPASS: Correctness, Modeling and Performance Of Aerospace Systems http://www.compass-toolset.org/

Cheddar: A resource scheduling analysis tool http://beru.univ-brest.fr/~singhoff/cheddar/

AADL Inspector: by Ellidiss Software www.ellidiss.com

ASIIST: real-time analysis Cyber Physical Systems Integration Lab

Ocarina: ENST. An AADL-based code generation tool suite available at http://aadl.enst.fr/ocarina/ & http://libre.adacore.com/tools/ocarina/

AADL & BIP: plug-in to interface AADL models with the Behavior Interaction theory (BIP) language http://www-verimag.imag.fr/Tools

Open Source AADL Analysis Tools - 2

Resolute: architectural assurance cases, integrated into OSATE Rockwell Collins

Agree: behavioral model checking, integrated into OSATE, Rockwell Collins

SysML to AADL Translator: integrated into OSATE, Rockwell Collins

Power Consumption Analysis Toolbox: integrated into OSATE, Lab-STICC developed under the SPICES project

EDICT Tool Suite: dependability analysis, WW Technology Group

Requirements Modeling Tool for AADL: by UBS/Lab-STICC. Available via Open-PEOPLE Open Power and Energy Optimization Platform and Estimator.

Additional information is available through the AADL Public Wiki

www.aadl.info/wiki

Large-Scale Development

Component evolution

- Component templates & refinement
- System families
- Component variants
- Components as extensions of other components
- Model configuration by property values

Large models & team development

- Components organized into AADL packages
- Public & private package sections
- Independently developed packages
- Version management of AADL packages
- Model integration

Outline: AADL Standard & ACVIP

- Challenges in embedded software systems
- Modeling-driven and architecture-centric engineering
- Overview of SAE AADL Standard suite
- AADL Language Overview
- AADL Tools
- Summary

Benefits

Model-based embedded system engineering benefits

Analyzable models drive development

Prediction of runtime characteristics at different fidelity

Bridge between control & software engineer

Prediction early and throughout lifecycle

Reduced integration & maintenance effort

Benefits of AADL as SAE standard

Common modeling notation across organizations

Single architecture model augmented with properties

Interchange & integration of architecture models

Tool interoperability & integrated engineering environments

END OF MODULE 1