Model-Based Systems Engineering and Multi-Paradigm Modeling and Simulation, two faces of the same coin.

Jerome Hugues

Softw are 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.

DM23-1051

BLUF: MPM vs. MBSE? Both!
(Let's start with the conclusion)

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)

"Multi-Paradigm Modeling (MPM) offers a foundational framework for gluing the several disciplines together in a consistent way. [..] MPM offers processes and tools that can combine, couple, and integrate each of the views that compose a system." [MPM4CPS CfP]

One (provocative) thought: why are we doing that?

Obviously: to design, implement, and assure system, we need the most suitable notations to define system elements and implement them, but also to extract and combine evidence

Two theses

1. *one should not do MPM without MBSE, as MBSE defines a master plan*
2. *one cannot do MBSE without MPM, as MPM provides the formal foundations*

Outline

1. Context
2. About AADL
3. An old story of Multi-Paradigm Modeling: The TASTE project
4. Process models for engineering: The TwinOps project
5. Process models for assurance: The ALISA DSLs
6. Wrapping-up

Overarching objectives – or what I am striving for

But is it MBSE or MPM ?

[….] assists engineers (mainly software, but also CPS and systems engineers) with models to

Concepts

Organize stakeholders needs and elicit requirements

Capture system elements – design, reverse engineering or COTS

- Interfaces, components internals (static and behavioral), and
- a system architecture built from those: deployment, (re-)configuration

Apply analytical frameworks to assess model's "compliance to some objectives"

- Syntactic, conformance to guidelines, patterns "well-formedness"
- Quality of system, w.r.t. performance, safety, security, behavior metrics
- Behavioral correctness, e.g. model checking, simulation, proofs

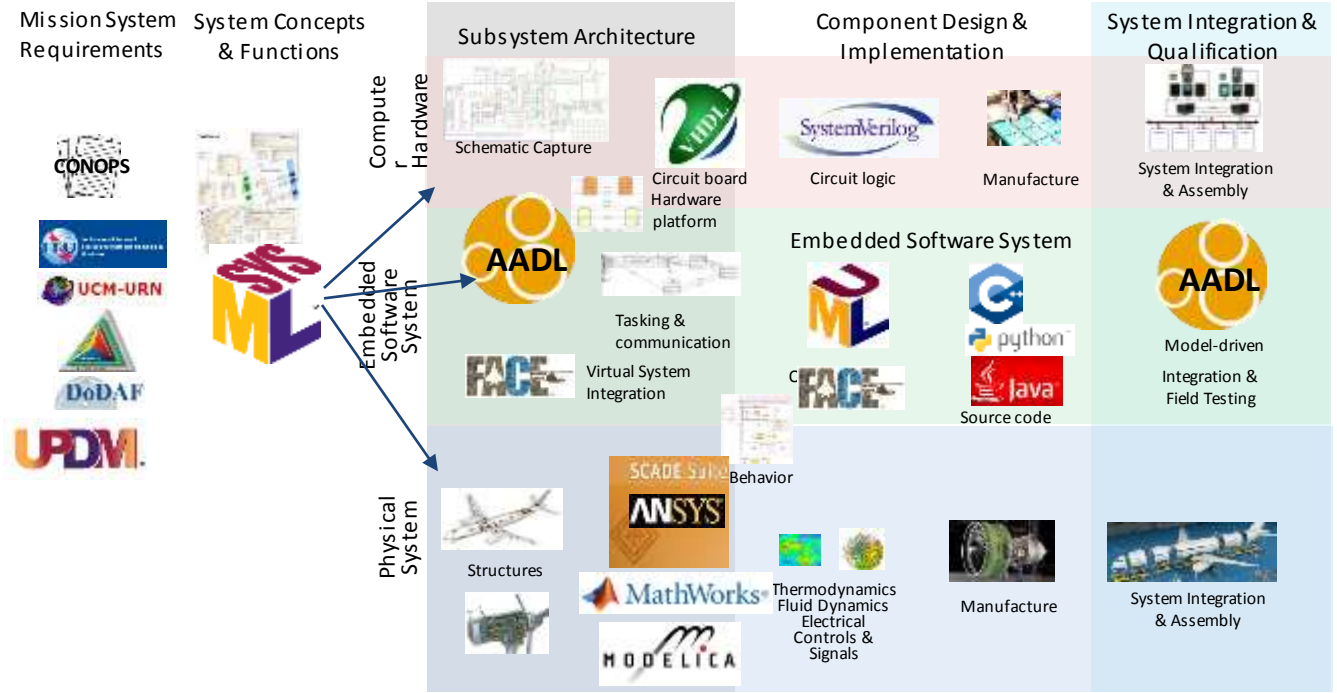
Models as processable artifacts to guide the software engineering process

System

- "Modeling is the new programming"

Provide more insights than CAD or code-only solution through relevant abstractions and automation

Modeling landscape – science removed for the faint of heart



ISO15288 – Processes for Systems Engineering

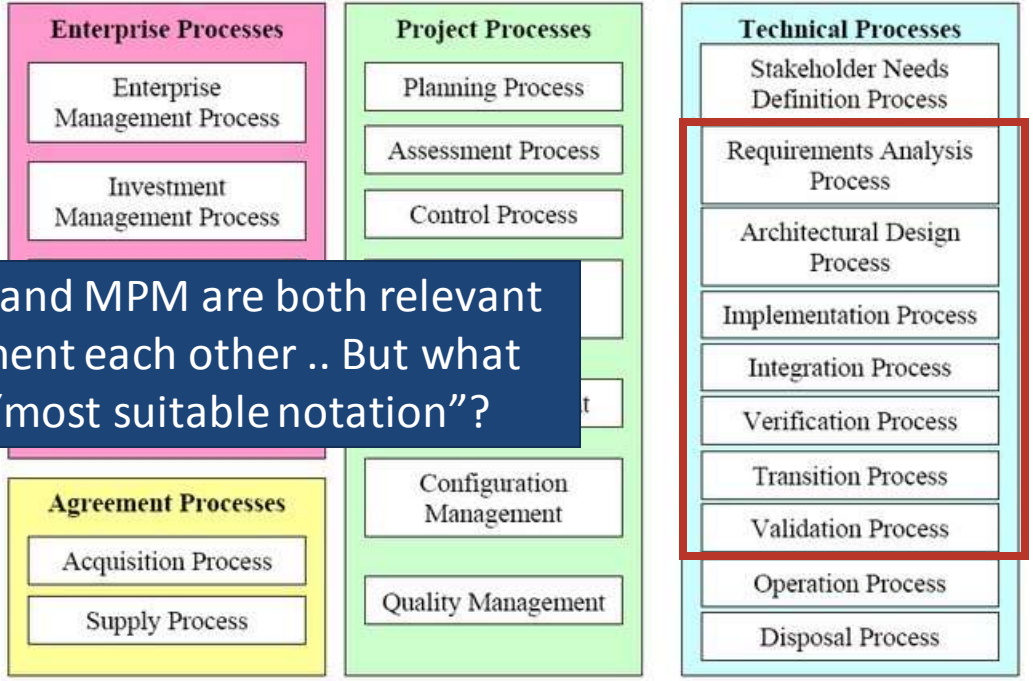
A process is a set of interrelated or interacting activities which transforms inputs into outputs (ISO 9000:2005)

Processes are organized so that they contribute to specific goals. They are structured by an *argume* justifies the existence of

MBSE covers (mostly) technical processes, but (may) lack many advanced V&V or simulation support

MPM provides solutions only for a subset .. But interconnected with all other process

Hence, MBSE and MPM are both relevant and compliment each other .. But what about the "most suitable notation"?



The system life cycle processes Source: ISO/IEC 15288, Figure D.8

ISO 15288 hass **23+processes** which have **123 outcomes** derived from **403 activities**.

MBSE – It's not just about SysML

SysML supports 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...

Claim: SysML supports MBSE, helps structuring top-most engineering artefacts through specific design methodologies (OOSem, Harmony methods, …)

We need another language with precise semantics for analysing CPS: AADL

Outline

1. Context
2. About AADL
3. An old story of Multi-Paradigm Modeling: The TASTE project
4. Process models for engineering: The TwinOps project
5. Process models for assurance: The ALISA DSLs
6. Wrapping-up

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

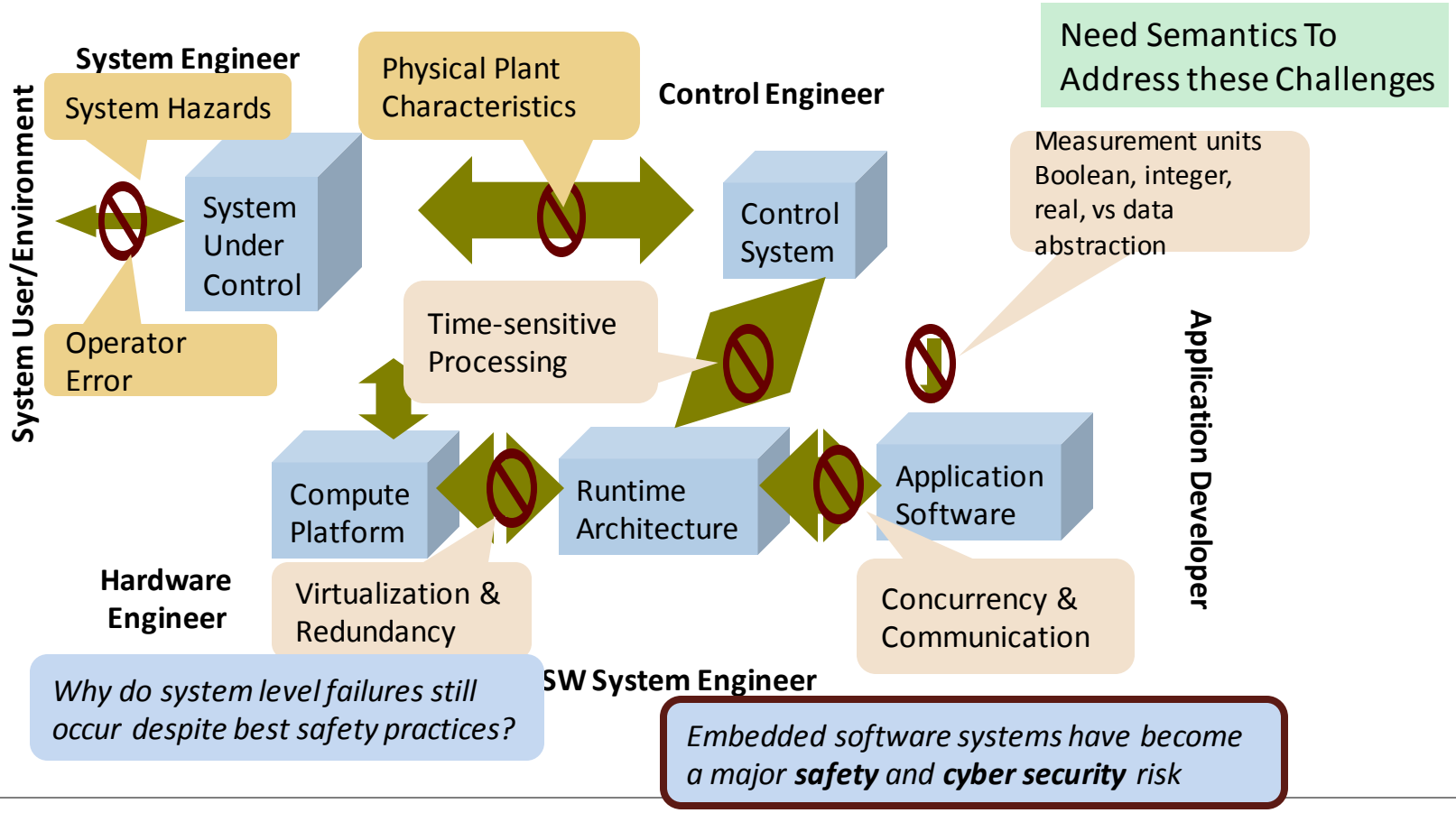
AADL Model-Based Engineering for Cyber-Physical Systems@SEI

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.

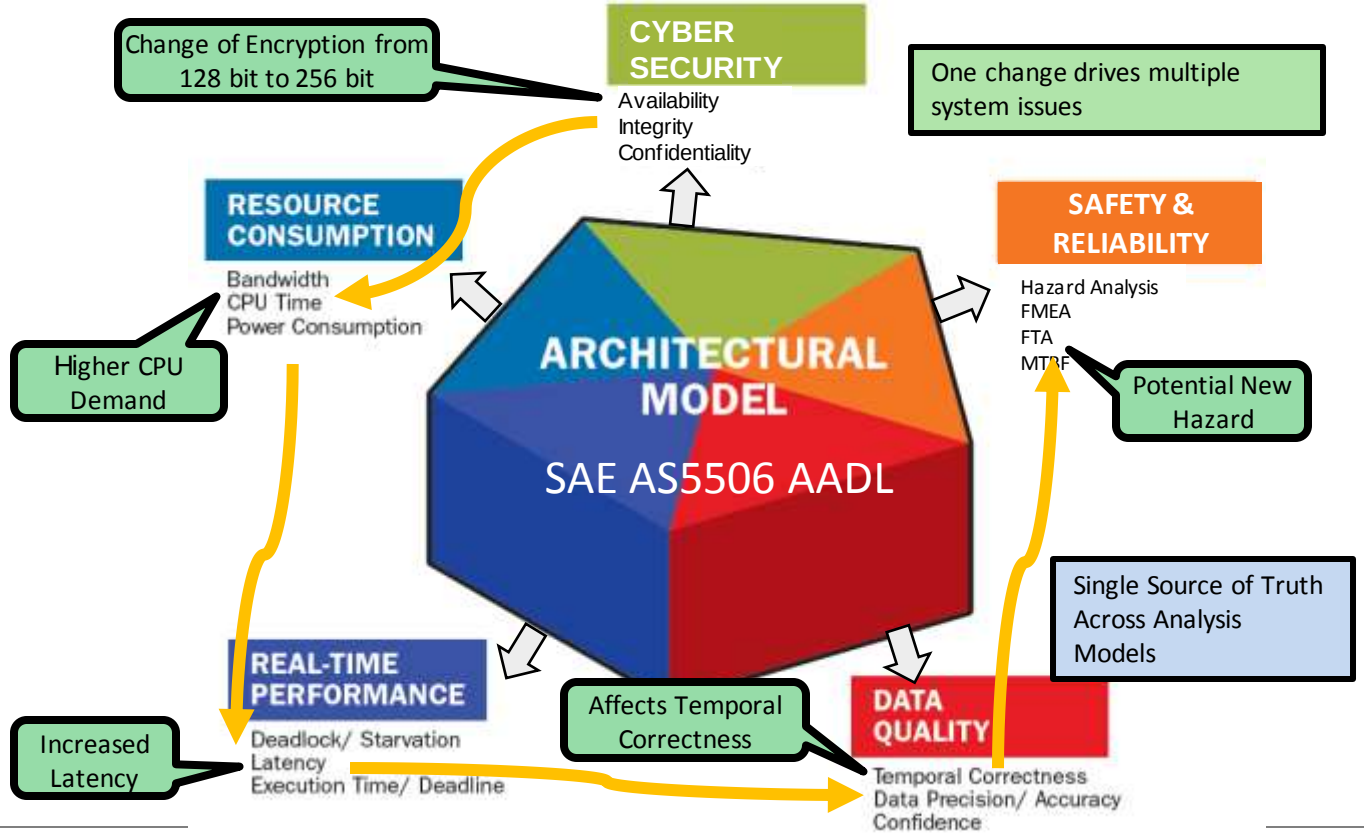
At the SEI, my team works on analysis techniques for safety-critical models

- Analysis operates either on the system, or a model of the system
- Models have different levels of fidelity:
  - Either ***an abstraction of an existing*** system
  - Or ***a blueprint of a future*** system
- Our main line of research is on AADL, an architecture description language
  - Scheduling or latency analysis, generation of fault trees, code generation, …
  - Each result from analysis is some form of an evidence
- We focus on both MPM and MBSE research questions
- We do support programs, we have end users with strong questions

Interleaved problems – why we need MPM-like techniques



One example of unrolling concerns



AADL Standard Suite (AS-5506 series)

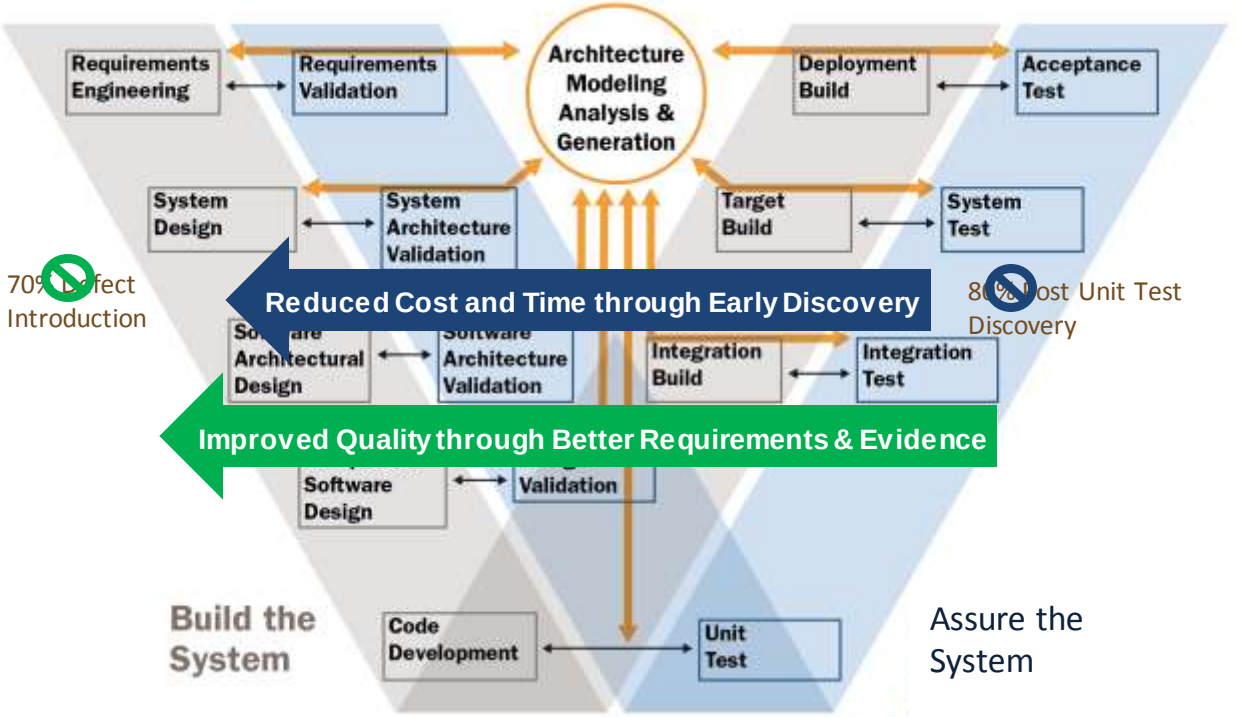
Core AADL language standard [V1 2004, V2 2012, V2.2 2017, V2.3 2022]

- Focused on *embedded software system modeling, analysis, and generation*

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]

- Evidence produced as a result of automated tool-supported analysis
  - Performance analysis: worst-case response time, schedulability of the system
  - Safety analysis: computing fault trees, probability of reaching an unsafe state
  - Automated model review: conformance to modeling guidelines
  - Code generation: generating “correct-by-construction” software

The Impact: Improved Cost, Time and Quality



AADL as an integration substrate for MPM

AADL support the precise design of CPS architecture as interconnected runtime elements (i.e., threads, processors, buses) , each contribute to system metrics evaluation: performance, safety. Natural connection with formal methods and code generation.

Non-exhaustive list of capabilities (1,600+ publications and counting):

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

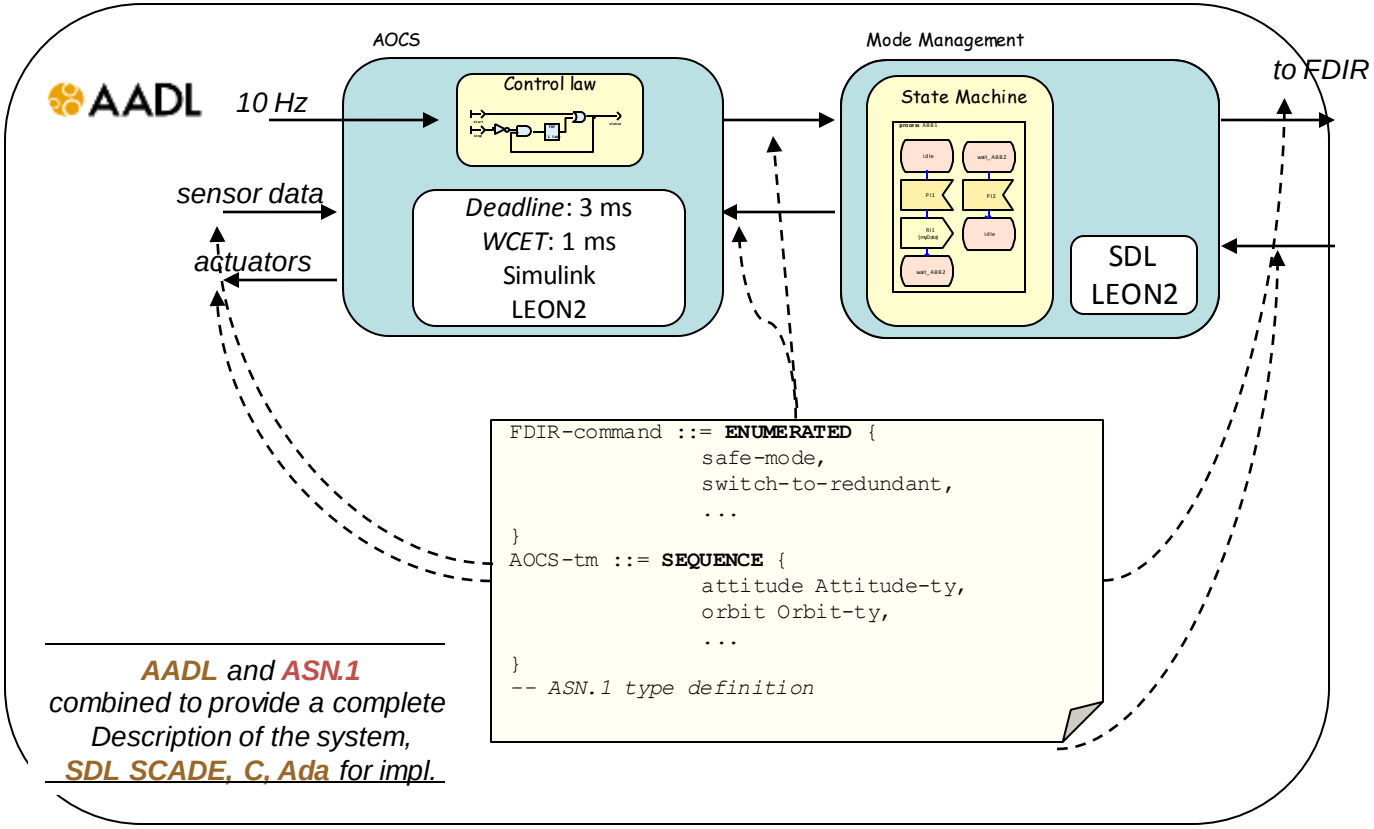
Performance evaluation: real-time and network calculus
Fault analysis: OSATE, COMPASS,
Mapping to Stochastic Petri Nets, PRISM
Security: CAMET (DoDI 8510.01)
Simulation: ADeS, Marzhin
Energy consumption of SoC: OpenPeople project
Code generation: SystemC, C, Ada, RTSJ, Lustre
WCET analysis: mapping to Bound-T

Claim: AADL is a contributor to MPM, helps connecting paradigms, either at design-time or analysis-time, ultimately for assuring systems

Outline

1. Context
2. About AADL
3. An old story of Multi-Paradigm Modeling: The TASTE project
4. Process models for engineering: The TwinOps project
5. Process models for assurance: The ALISA DSLs
6. Wrapping-up

Why multiple modeling notations? A story by ESA



What is the TASTE process?

https://taste.tools/

TASTE deriv**es from ASSERT** (FP6, 2004 - 2008)

The TASTE process is based on simple observations

- a system – ANY system – is made of ***heterogeneous components***, that have to live and communicate together
- system builders have other concerns than ***software implementation*** *details*,
- and good software engineers are ***unhappy*** *when they have* ***"just" to develop*** application code and follow associated process: their skills are misused, better used to hack-n-play: add drivers, play with hardware, add verification/testing facilities, configure RTOS, etc.

Some considerations on Model-Based S(oftware|ystem) Engineering

- Desired functionality have reached levels (dangerously) close to being unmanageable by humans. "Empirical" methodologies have reached their limit...
- Model Driven Engineering offers some notion of correctness, at the model level
- Tools generate C/Ada code – but no established standard for interoperability
- Messages/data exchanged face the same issue – no common data modeling
- Some options for automation – remove human from the loop

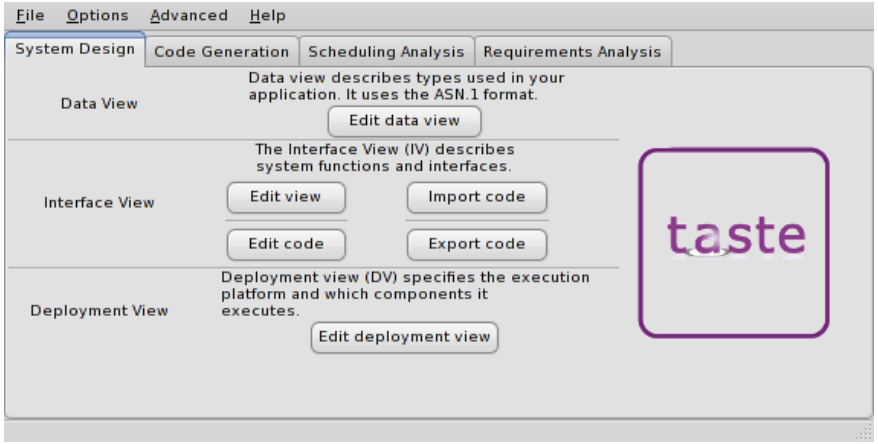
The TASTE process proposes solutions to

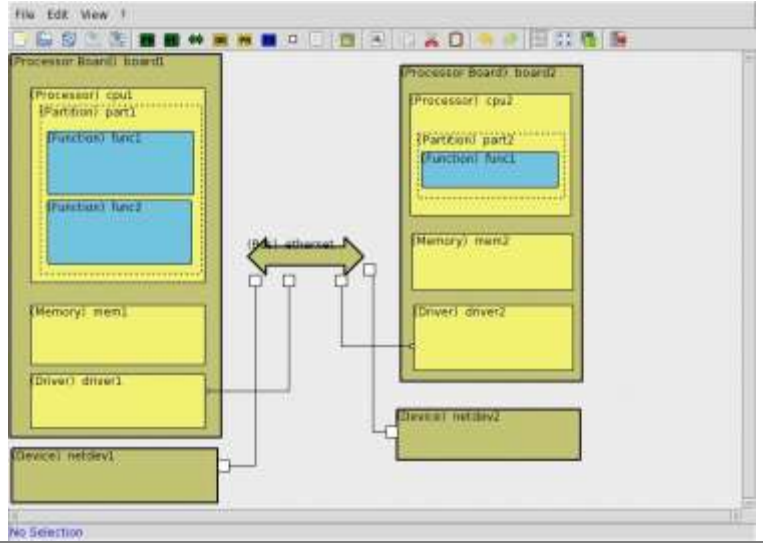
- capture a system using user-friendly (yet formal) modeling techniques
- automate repetitive and error-prone software activities
- build an homogeneous system having heterogeneous components

Modeling your system

TASTE proposes an integrated set of tools for modeling

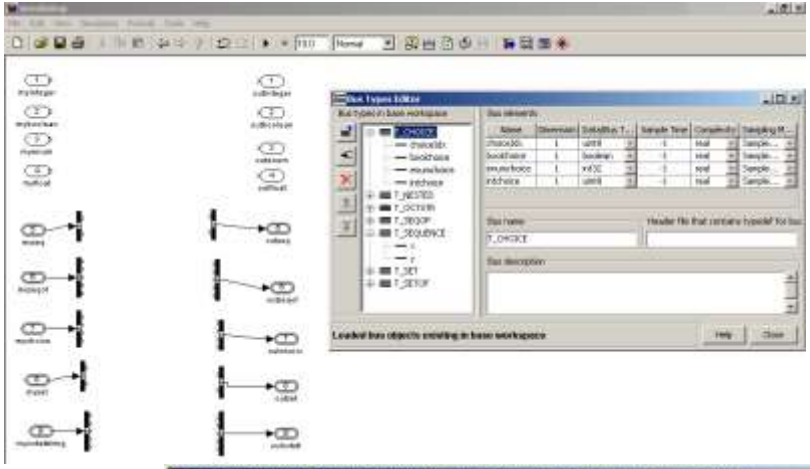
- Graphical DSL, provides basic blocks + semantics
  - Follow Ravenscar computational model
- Or use directly AADL, and connect it with your own process

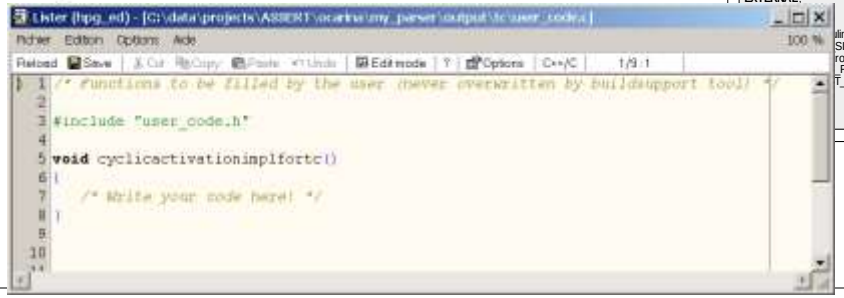




Adding function behavior: pick your notation

Generation of code “skeletons” from AADL interfaces





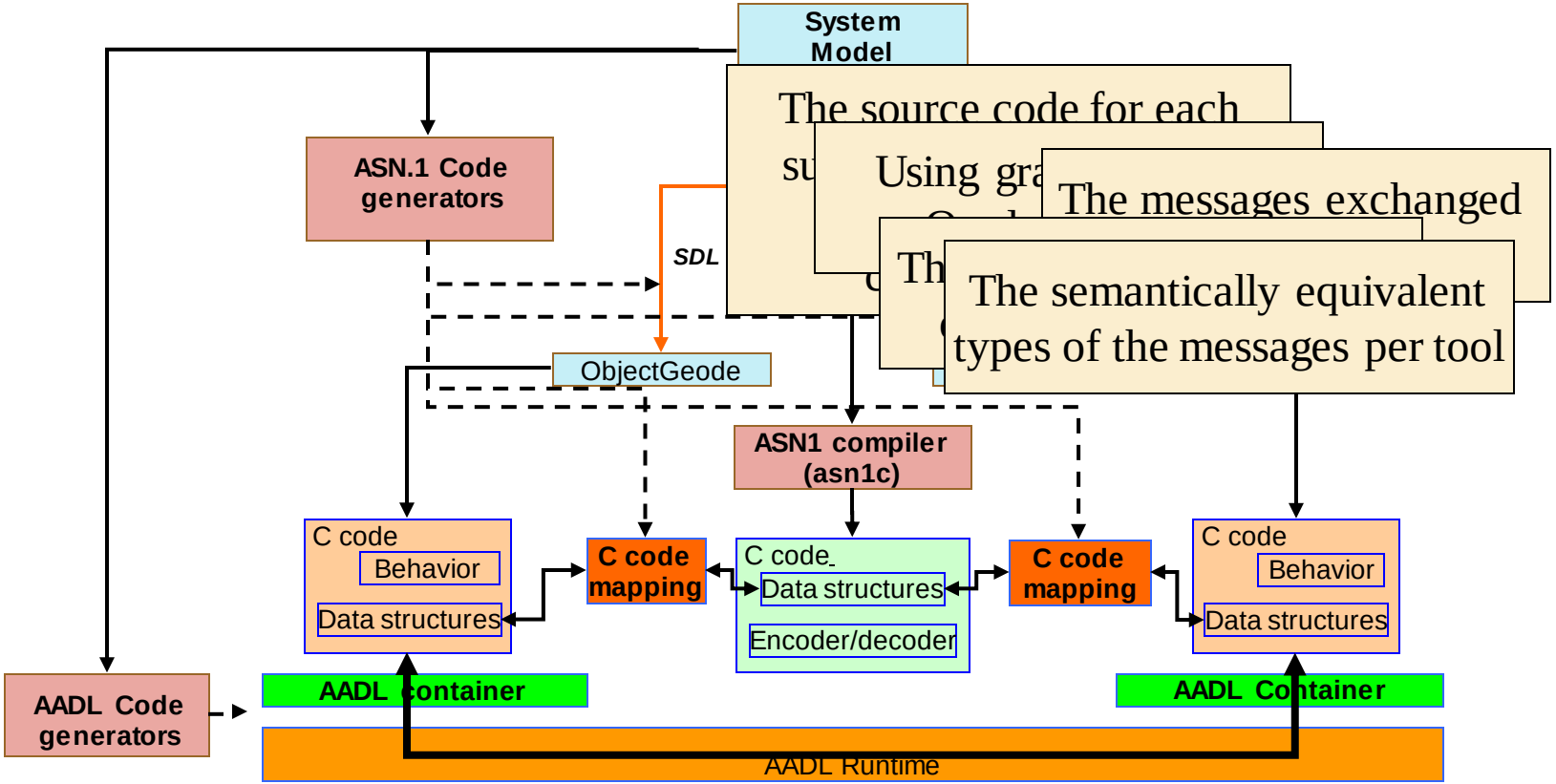

```
system basic_fv
  USE Datamodel;
  SIGNAL basictotc (T_TM);
  SIGNAL tcommand (T_HLTC_PLUS);
  SIGNAL basictocontrol (T_CONTROL_IN);
  SIGNAL controldow ntobasic (T_CONTROL_DOWN_OUT);
  SIGNAL controluptobasic (T_CONTROL_UP_OUT);
  SIGNAL cyclicactivationimplementation;

procedure aplc_basic_op COMMENT '#c_predef';FPAR
  IN thrusters_opening T_THRUSTERS_OPENING,
  IN pfs_iw m_arming_relay_status_on T_PFS_IWM_ARMING_RELAY_STATUS_ON,
  IN pfs_hltc_red_button_is_on T_PFS_HLTC_RED_BUTTON_IS_ON,
  IN msu_id T_MSU_ID,
  IN pfs_ew m_msuy_msux_hs T_PFS_EWM_MSU_MSU_HS,
  IN ftcp_health_status T_FTCP_HEALTH_STATUS,
  IN pfs_ew m_dtg12_msu T_PFS_EWM_DTG12_MSU,
  IN hltc T_HLTC,
  IN end_boost_is_reached T_END_BOOST_IS_REACHED,
  IN sun_is_aimed T_SUN_IS_AIMED,
  IN/OUT pfs_ew c_msu_pde_t T_PFS_EWC_MSU_PDE_T,
  IN/OUT pde_cmd_a T_PDE_CMD_A,
  IN/OUT dpu_cmd T_DPU_CMD,
  IN/OUT set_pfs_ew c_msu_dtg_mode_coarse T_ON_OFF_CMD,
  IN/OUT hltm T_HLTM,
  IN/OUT pfs_ew m_msux_msuy_hs T_PFS_EWM_MSU_MSU_HS,
  IN/OUT cam_mode T_CAM_MODE,
  IN/OUT controller_to_be_activated T_CONTROLLER_TO_BE_ACTIVATED,
  IN/OUT navigation_output T_NAVIGATION_OUTPUT;
EXTERNAL;

ink COMMENT '#c_predef';FPAR
SIMULINK_IN,
rol_in,
_FOR_SIMULINK_OUT,
T_Control_in;
```

basic_fv

Implementation of the TASTE process

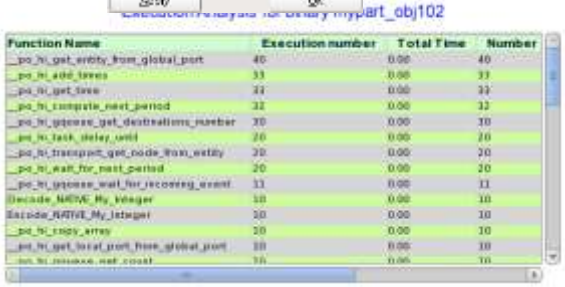


Gathering metrics from projects

Scheduling, memory, coverage analysis

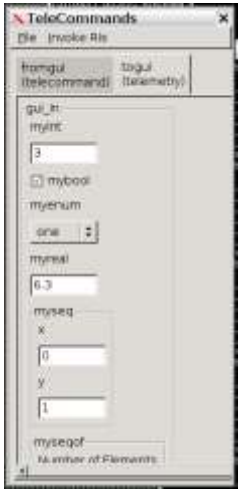
- Integrate third-party tools, consolidate results at project level

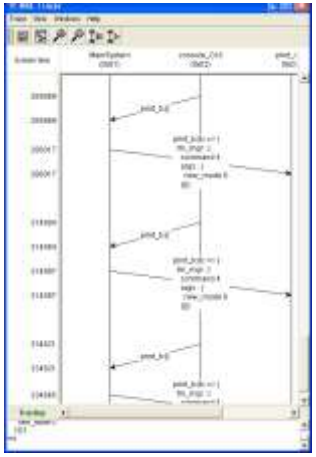
All available from main GUI, or command line

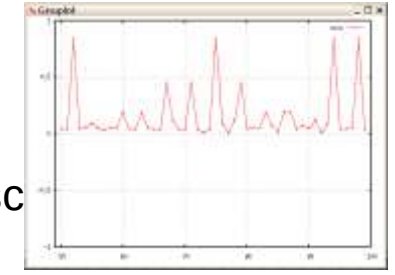


Testing and so forth

Rapid prototyping:
the toolchain generates GUIs to quickly test the system, replay capabilities using MSCs







Simulation and Analysis:
Data can be monitored using real-time plotting.

Documentation: ICDs are generated automatically with a desc

data binary encodings (ASN.1 uPER Encodings)

```
MY-MODULE DEFINITIONS ::= BEGIN

MySequence ::= SEQUENCE {
   field1          INTEGER (5..4294967295),
   field2          INTEGER (5..4096) OPTIONAL,
   field3          BOOLEAN ,
   field4          MyChoice,
```

MySequence (SEQUENCE)					
Sequence preamble		Bit mask		2	2
1	field1	INTEGER	No	32	32
2	field2	INTEGER	Yes	12	12
3	field3	BOOLEAN	No	1	1
4	field4	MyChoice	No	3	162
5	field5	OCTECT STRING	No	8	∞
6	field6	MySequenceOf	Yes	16	1207

Lessons learnt#1: Pragmatic MPM is ”easy”, but ..

TASTE demonstrators: FDIR, Galileo receiver, robotic arm, …

Focus on supporting most activities in V cycle, using AADL as integration substrate

- Prototyping, testing, V&V activities (scheduling, resources) at both model and code levels
- (ongoing) Support for formal methods

Pragmatic/opportunistic multi-paradigm: *the most relevant notation at each step!*

Models and processing steps align with established practice in the space domain

- Peek standard, most suitable, notations for each design, coding, and analysis activity
- Remove manual work in critical paths through aggressive model transformations
- Orchestration of **both** modelling and build processes is key **-> *hidden master plan***

One instance of MPM, but process is **NOT defined by a model,** can we do better?

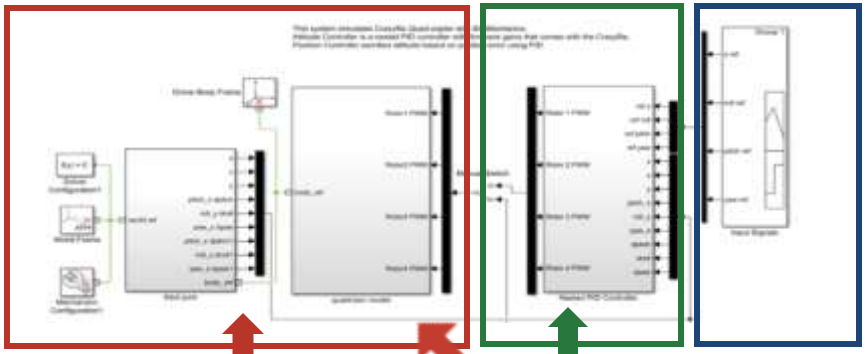
Outline

1. Context
2. About AADL
3. An old story of Multi-Paradigm Modeling: The TASTE project
4. Process models for engineering: The TwinOps project
5. Process models for assurance: The ALISA DSLs
6. Wrapping-up

TwinOps problem space: CPS Integration and Testing

See [MPM4CPS'20] for details

High-Level Architecture

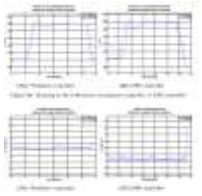


TwinOps: leverage other source of truth (e.g., CAD, Physics) to improve SW V&V

⇒ Use precise models instead of (naïve) abstractions for improved SW V&V

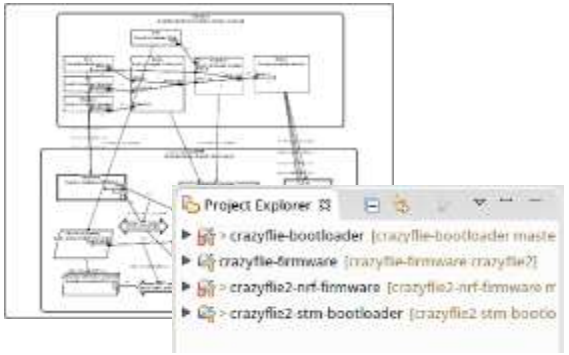
⇒ Combine domains, including SysEng

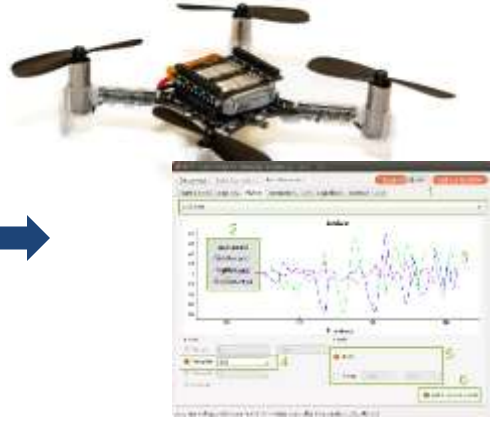
Implementation Space



+

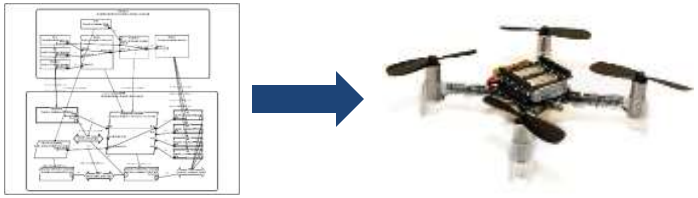
SW/HW Architecture





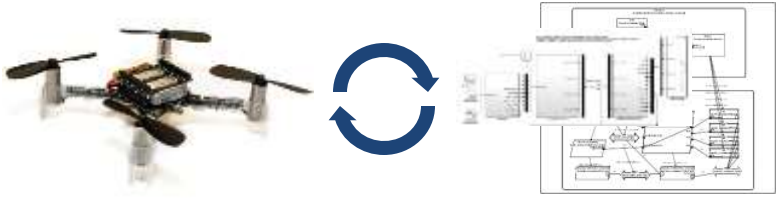
Technology Focus: Models and Code Generation

One can ***generate code*** from models ready to be embedded in the system (e.g., AADL to C) and get insights from the system to refine the model metrics.

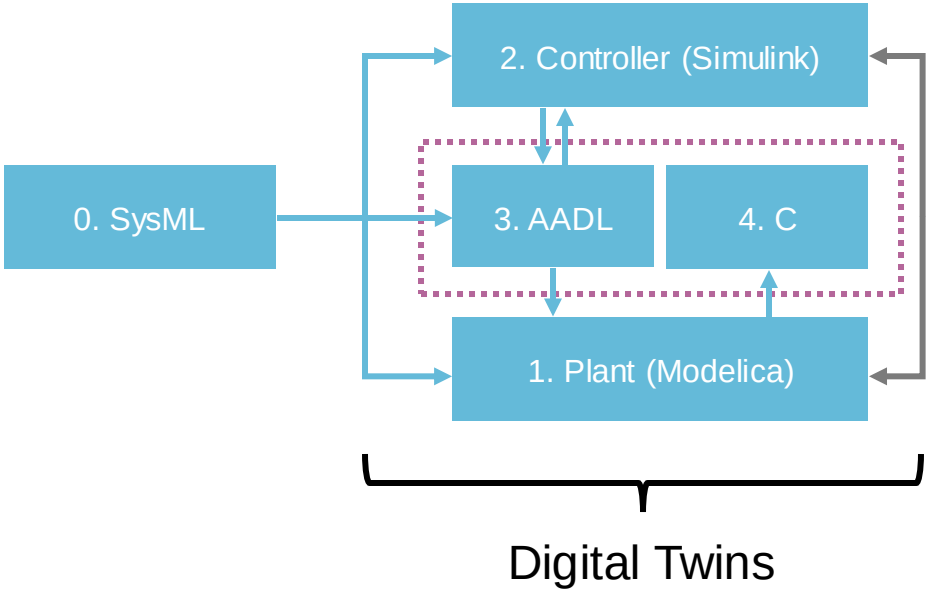


One can ***simulate models*** and generate simulation code as a mock-up of some system parts.

One can build ***Digital Twins***, that compare actual system and its digital simulated doppelganger.



MBSE and MPM at work



1-2-3-4: “mega-modeling” V&V
- 1-2: HLR validation
- 2-(3+4): validation of LLR
  1+(3+4): virtual integration

Digital Twins of UAV vs. UAV flying: validation of Modelica model, efficiency of the controller (overshoot verification) and timing verification of software.

MBSE and MPM at work

1-2-3-4: “mega-modeling” V&V
- 1-2: HLR validation
- 2-(3+4): validation of LLR

How to organize models?

AADL extends/refines the SysML model block diagrams

Simulink implements requirements for a controller

Modelica implements operational context

[Caveats not discussed here: a model refining another model should preserve some properties (structural, behavioral). Think Liskov, A/G contracts, etc.]

Notional process hidden: “$model_1$ + operation => $model_2$”

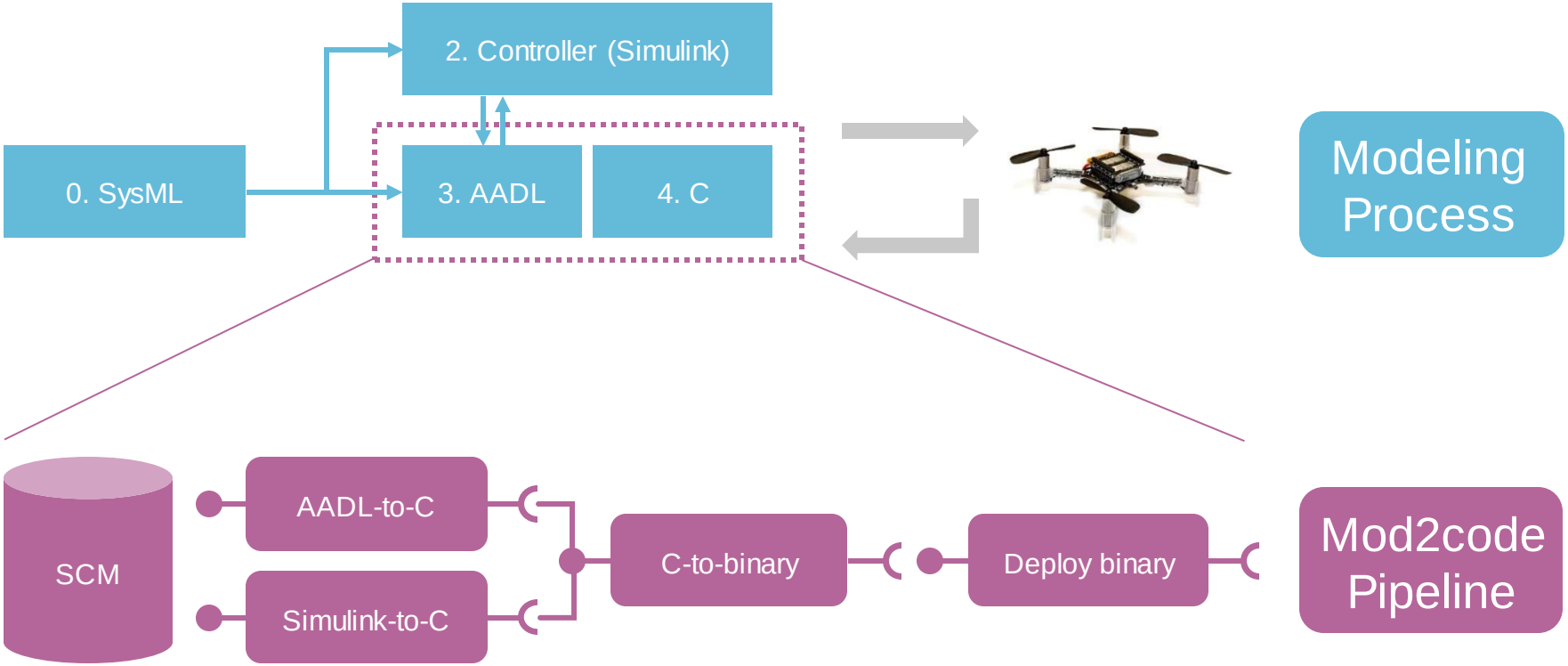
MPM must build on Systems Engineering and MBSE to define

Each process involved, specialization of ISO15288 processes

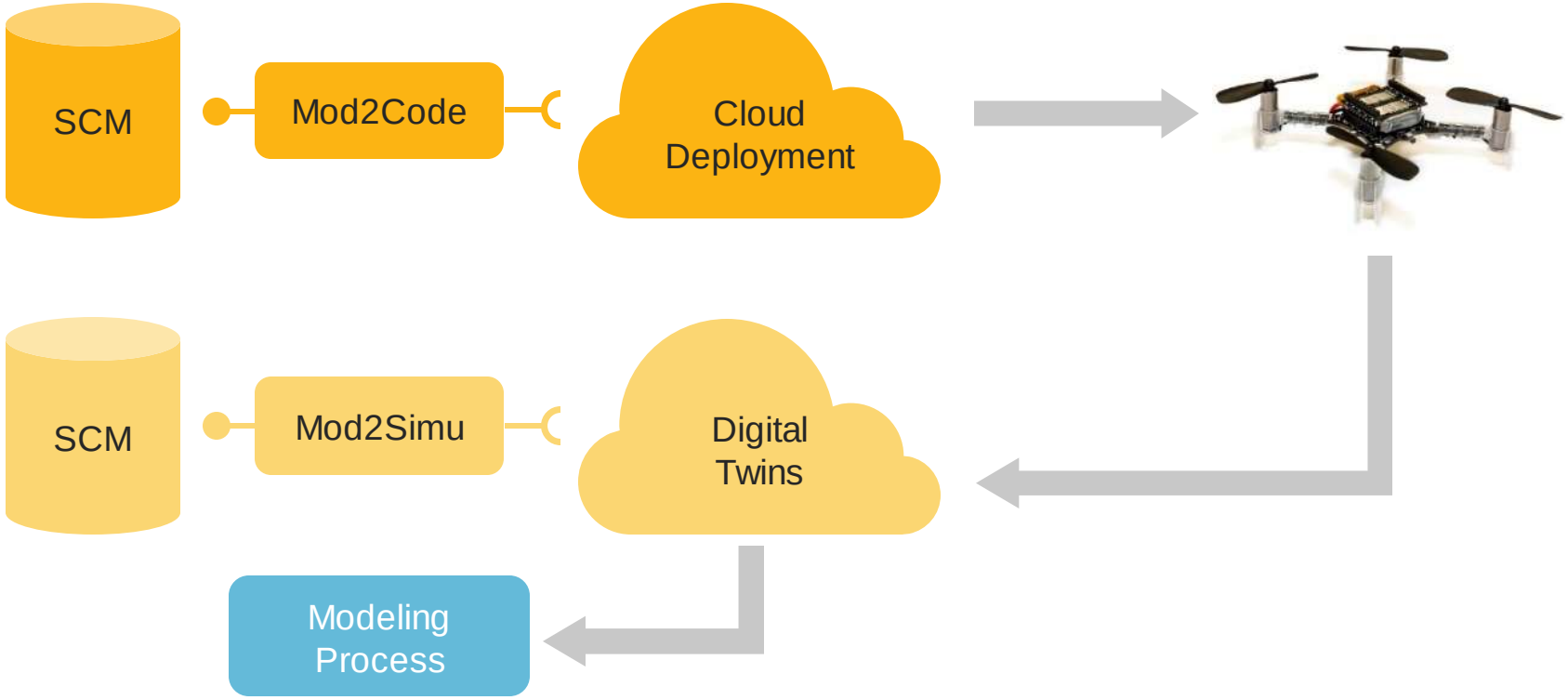
Goal of each step within a process

timing verification of software.

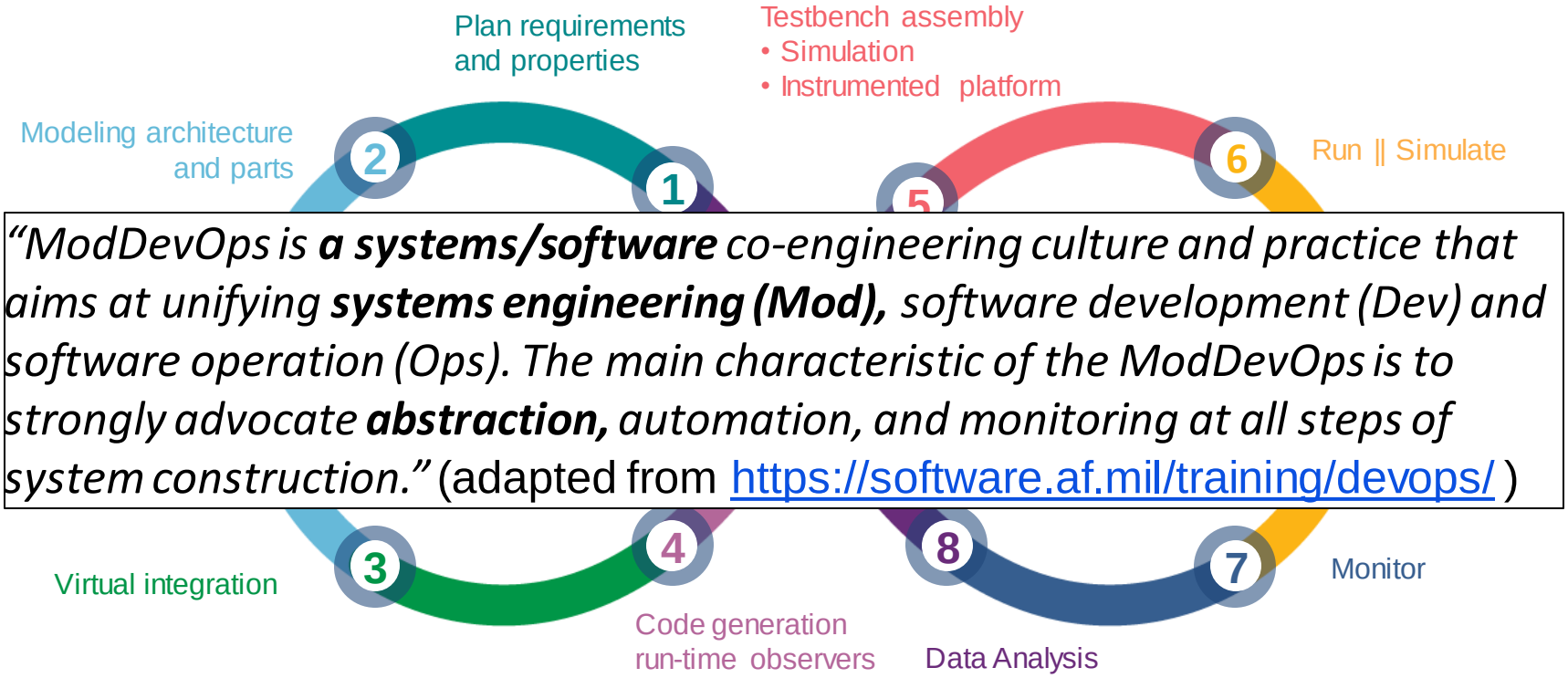
Orchestrating activities as “DevOps pipelines”



DevOps as a metaphor for process execution



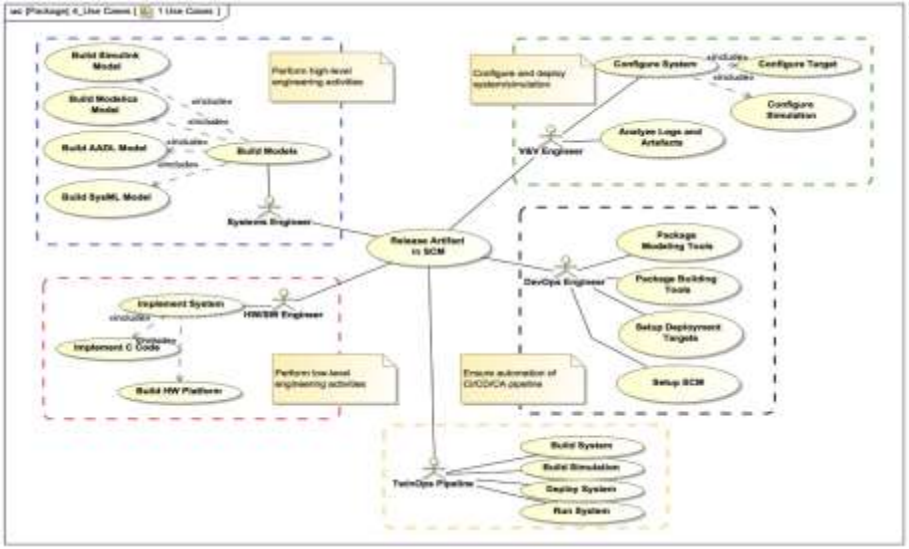
TwinOps: Continuous System Improvement through ModDevOps and Digital Twins

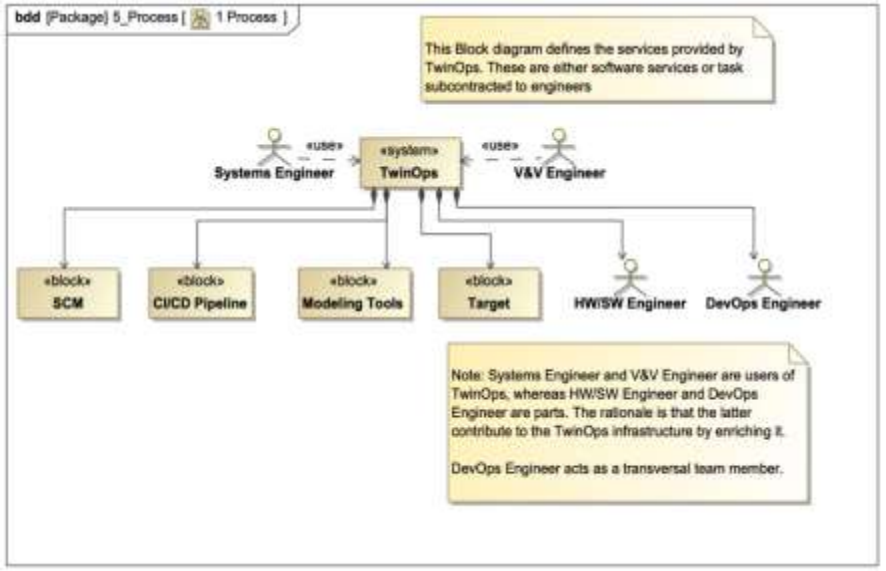


Process models – a.k.a. SysML strikes back

SysML **and** AADL provide foundations for architecting the system

SysML provides foundations for architecting process models (or BPMN, …)





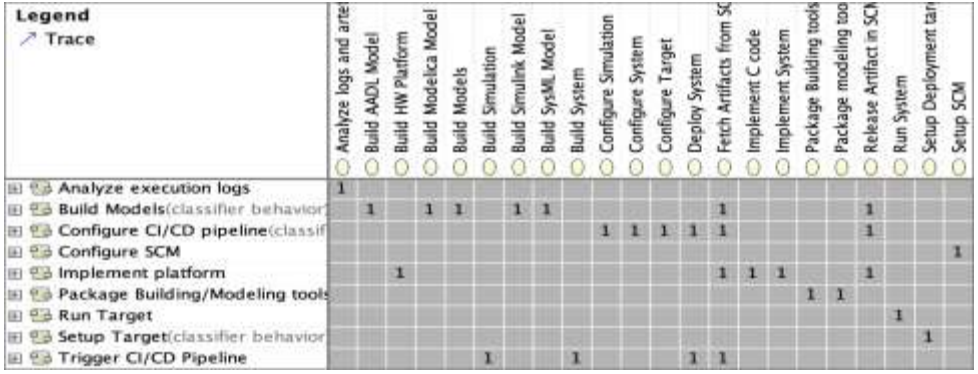
Process models – a.k.a. SysML strikes back /2

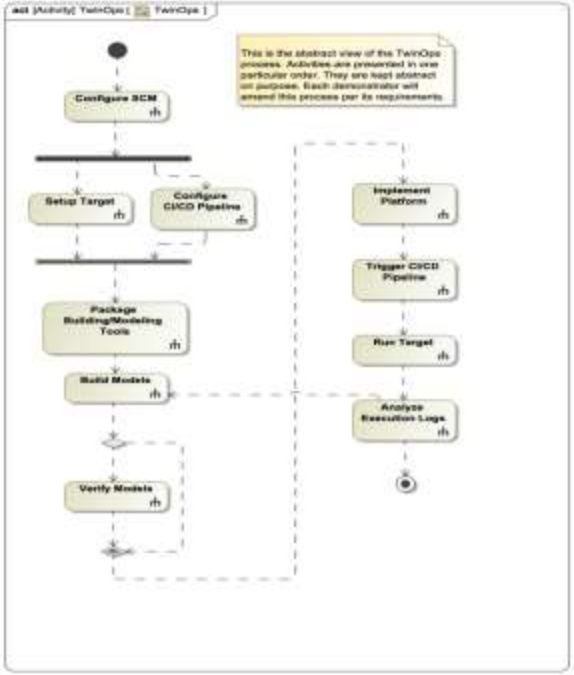
Making each step of the process model a visible artefact

- Generic steps + instantiation, e.g., “generate code from X”

Actual execution using typical CI/CD pipelines

- For provisioning tools, e.g., Modelica/OSATE container
- For automating simulation, analysis, code generation and build, and deployment on target, etc.





Lessons learnt #2: Efficient MPM using MBSE practice

We are **NOT** "herding cats", TwinOps shows how to rigorously define model-supported process that combine multiple "most suitable" models

From pragmatic to efficient MPM using MBSE abstractions

=> Make visible the process of combining models and model processing

Hierarchy of models is key to distinguish

system definition models from system engineering process models

Are we done yet? (time check: how far are we from the coffee break ;-)

No .. We are still **only** addressing **the left side of the V-cycle**

Outline

1. Context
2. About AADL
3. An old story of Multi-Paradigm Modeling: The TASTE project
4. Process models for engineering: The TwinOps project
5. Process models for assurance: The ALISA DSLs
6. Wrapping-up

Assurance as the final goalpost

We do models to build **assured** systems, i.e.,

As in properly defined, using proper engineering, with properly asserted properties

Many standards use requirements engineering to derive “what the system should do”

Requirements for a Patient Therapy System

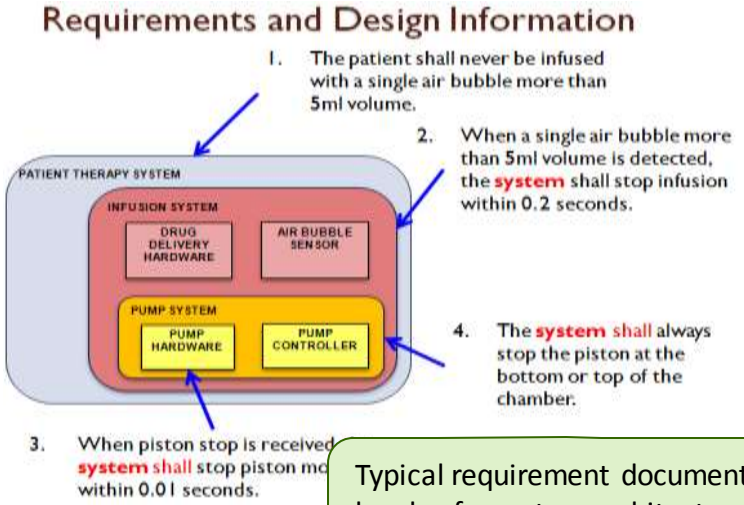
The patient shall never be infused with a single air bubble more than 5ml volume.

When a single air bubble more than 5ml volume is detected, the system shall stop infusion within 0.2 seconds.

When piston stop is received, the system shall stop piston movement within 0.01 seconds.

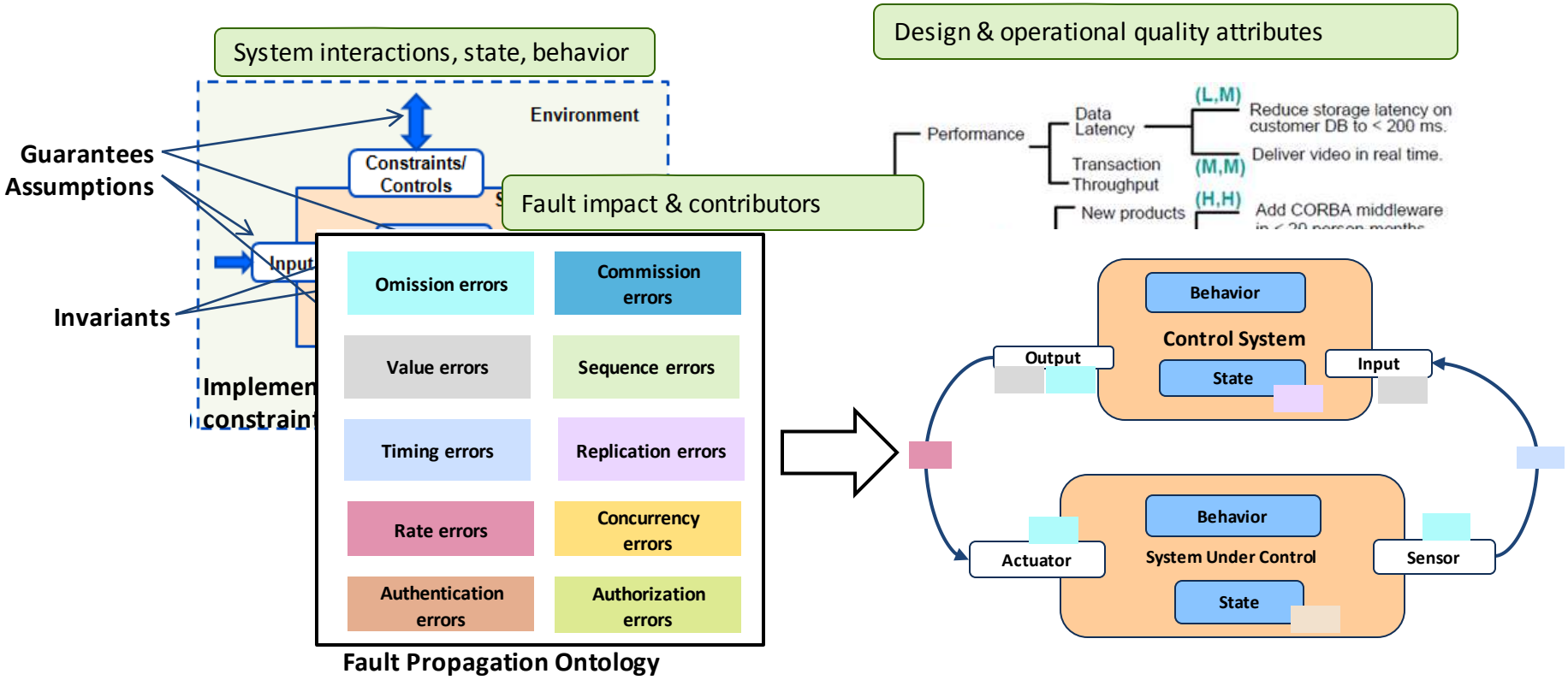
The system shall always stop the piston at the bottom or top of the

Importance of understanding system boundary.
Multiple layers of system boundary.



Typical requirement documents span multiple levels of a system architecture
Requirements specified a partial architecture

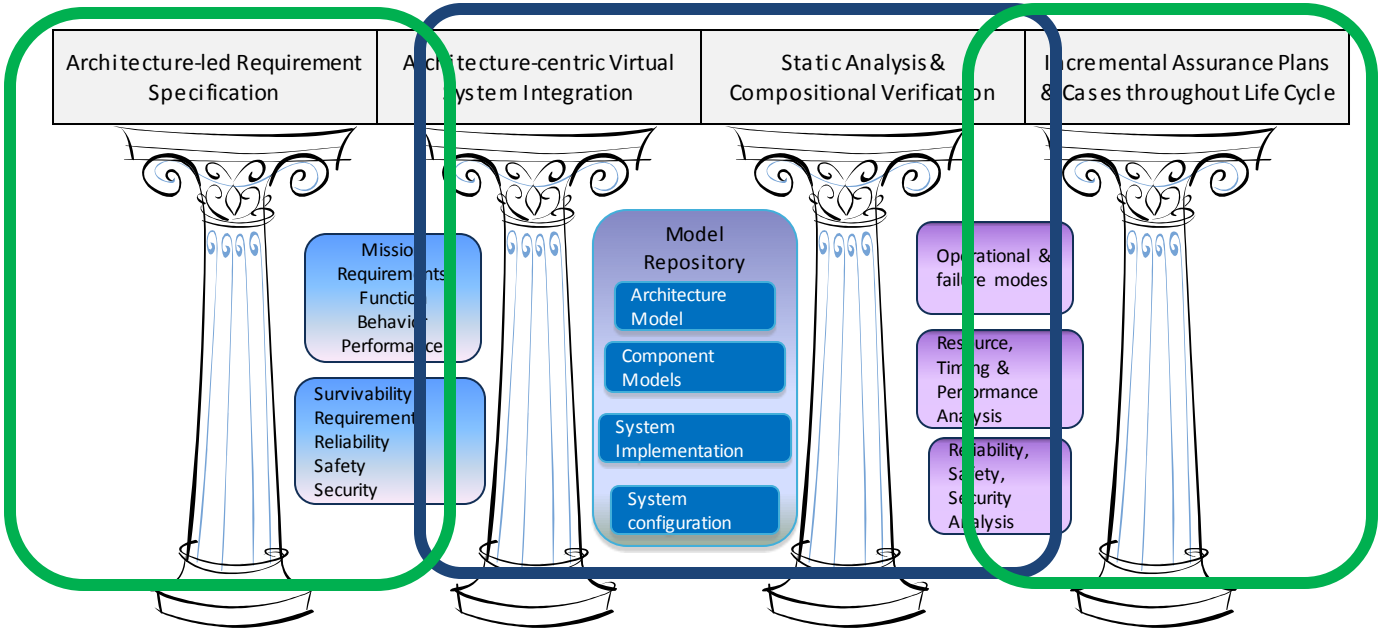
Three Dimensions of Requirement Coverage
All supported by AADL



Assurance & Qualification Improvement Strategy

Assurance: Sufficient evidence that a system implementation meets system requirements

2010 SEI Study for AMRDEC Aviation Engineering Directorate



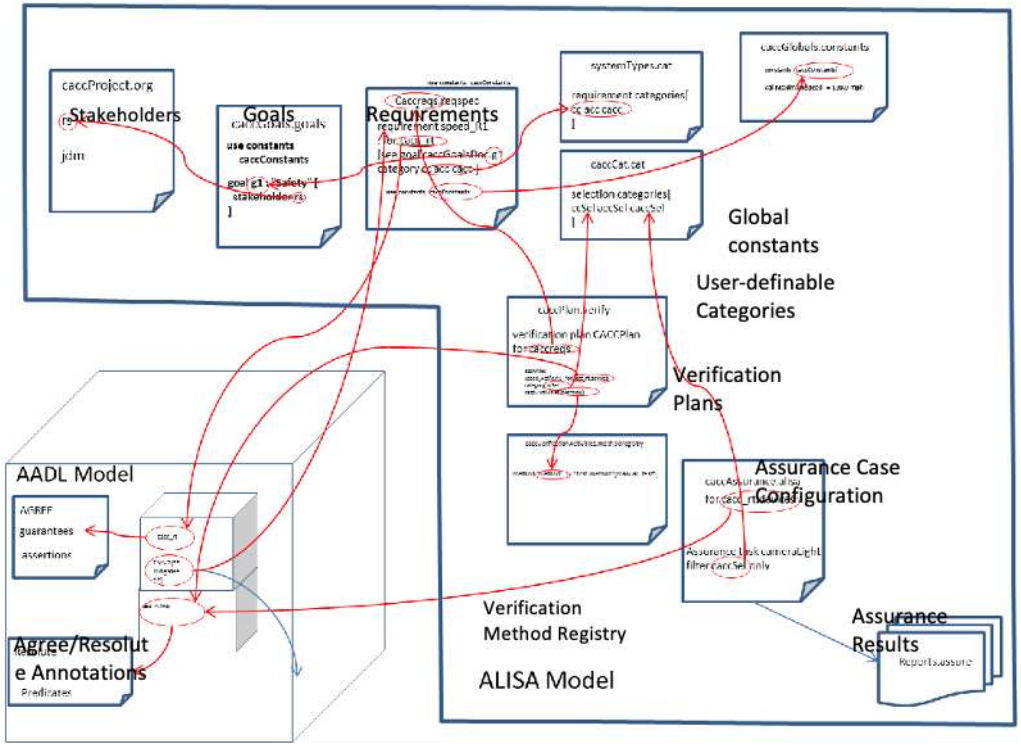
Architecture-Centric Virtual Integration (ACVIP) Architecture-led Incremental System Assurance (ALISA)

About the ALISA DSL

The ALISA DSLs provide for establishing links between

- the requirements written in ReqSpec and the AADL model
- the verification plans written in Verify

Verification methods are either written using Resolute (static, ≈ FOL), AGREE (dynamic, ≈ model checking), or OSATE plugins (e.g. FTA, latency)

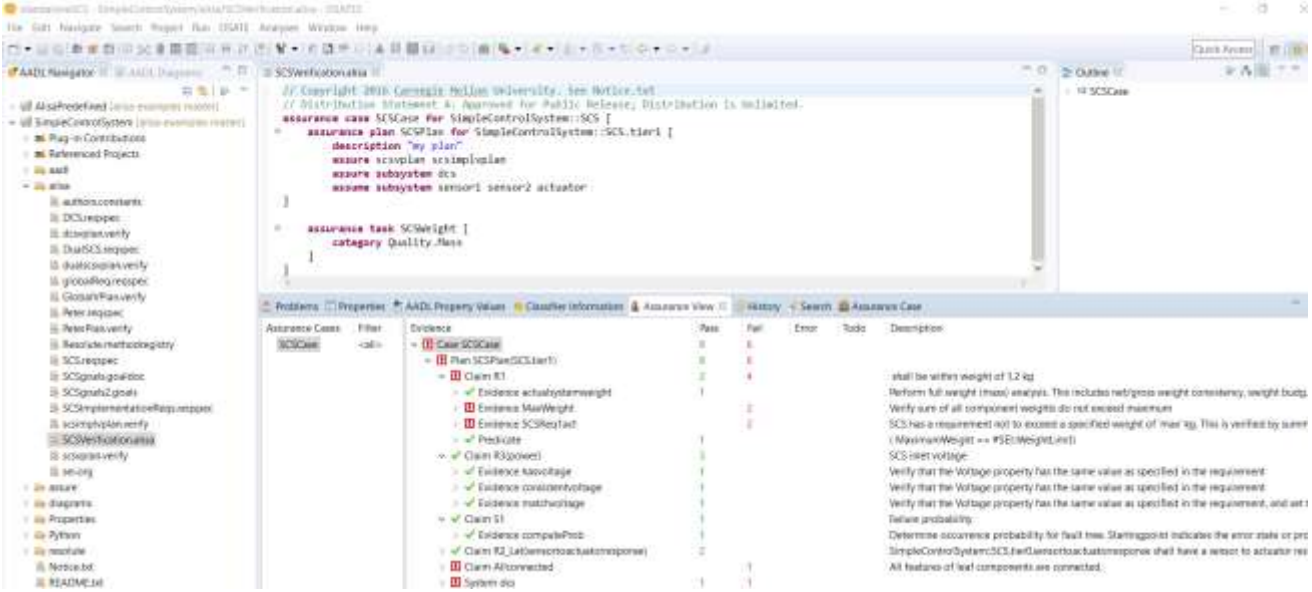


Assurance Case Configuration & Results

Why GUI also matters for both MBSE and MBE!

Executing a verification plan provides an **argument,** stating how requirements are met

- A verification plan is one form of process model: the process of assuring the system
- Presenting results either in OSATE GUI or using GSN



Lessons learnt #3: What is a “most suitable notation”? A never-ending story? A not finished one for sure

Efficient MPM and MBSE cannot rely on a user clicking at the right place at the right time

Typical issues include

- Metrics to assert a model is ready for processing: coverage? Rule-based?
- Conditions to assert that a model processing will yield a correct results (A/G contracts?)

Guiding process models’ execution is key for efficiency and correctness

Assurance models are another set of notations for improving MPM/MBSE effectiveness

- ≈ trace of the execution of a process model that connects outcomes of modeling or verification activities: ”*A is safe because property X is met as shown in evidence E”.*
- Comes with a formal semantics and graphical representation as well (GSN, CAE, …)
- Can be both validated and reviewed

Outline

1. Context
2. About AADL
3. An old story of Multi-Paradigm Modeling: The TASTE project
4. Process models for engineering: The TwinOps project
5. Process models for assurance: The ALISA DSLs
6. Wrapping-up

So, what is a “most suitable notation”?

Going back to the most meaningful definition of what models are about (IMHO)

> ***“To an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A.*** *The model relation is inherently ternary. Any attempt to suppress the role of the intentions of the investigator B leads to circular definitions or to ambiguities about "essential features" and the like.”*
> Minsky, “Matter, Minds, and Models”

Most suitable notation is not just about models for “asserting Y about X”, it is <u>also</u> about

- showing how X has been built (models and process models),
- how facts about X are established (process models)
- how those facts are combined for assuring X (assurance model)

In other words: what the relevant questions to be answered are, who the stakeholders are

Conclusion

Just opening the box: MBSE and MPM are strongly coupled

System models bring confidence for design, V&V, but ”master plan” missing

Process models to further mature MPM: increase automation, correctness

engineering dimension: support the construction of large complex systems

With “enough” confidence, models are useful.

Confidence you are doing the right model, using the right processing, getting the right result

Confidence it is worth the effort, and that “you’re not feeding the beast”

Open research questions

1. Define model “value” in a non-ambiguous way, as a complement to model language and semantics
2. Define “fitness” of a process model w.r.t. design goal .. so that we pick the “most useful notation”