A programming language view to model-driven engineering

Jérôme Hugues, ISAE/DMIA

jerome.hugues@isae.fr

Outline

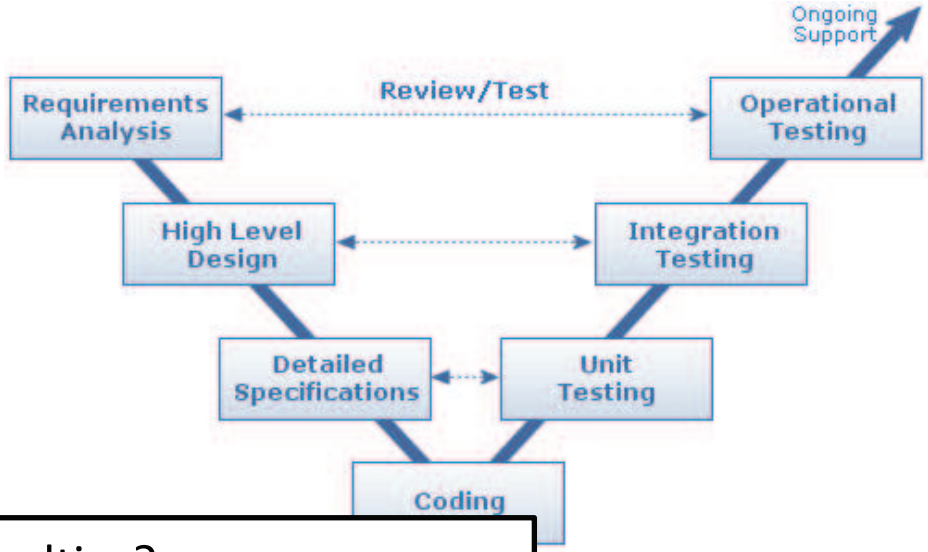
- Model-Based System/Software Engineering vs. the real world
- AADL, an overview
- MBSE as an extension to programming in the large

Outline

- **Model-Based System/Software Engineering vs. the real world**
- AADL, an overview
- MBSE as an extension to programming in the large

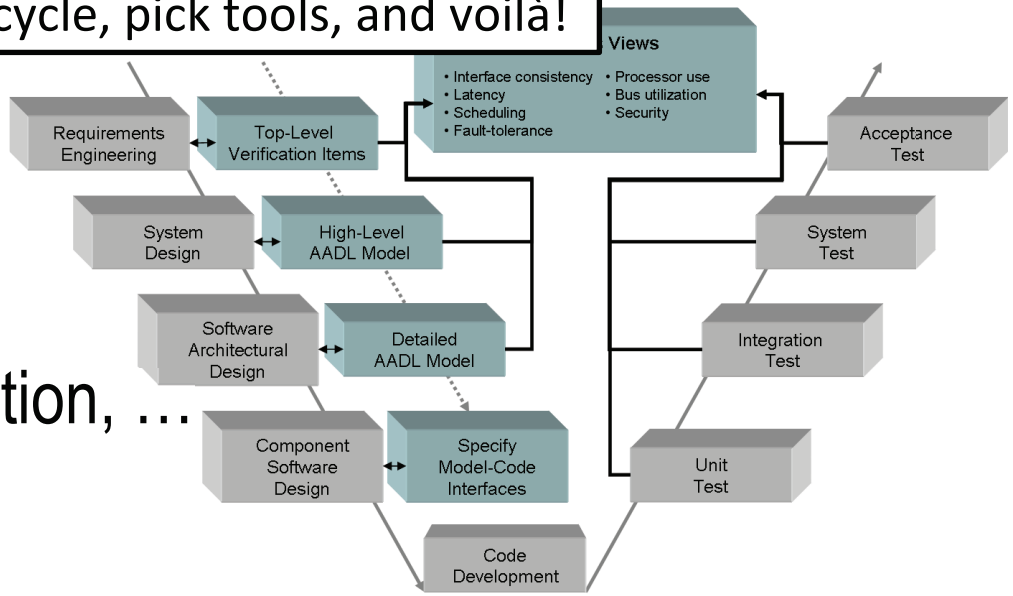
Engineering cycles

- **Now typical in SW engineering**
  - » Various refinements
  - + traceability across layers
  - + split across teams (HW, SW, …)
  - and consolidation

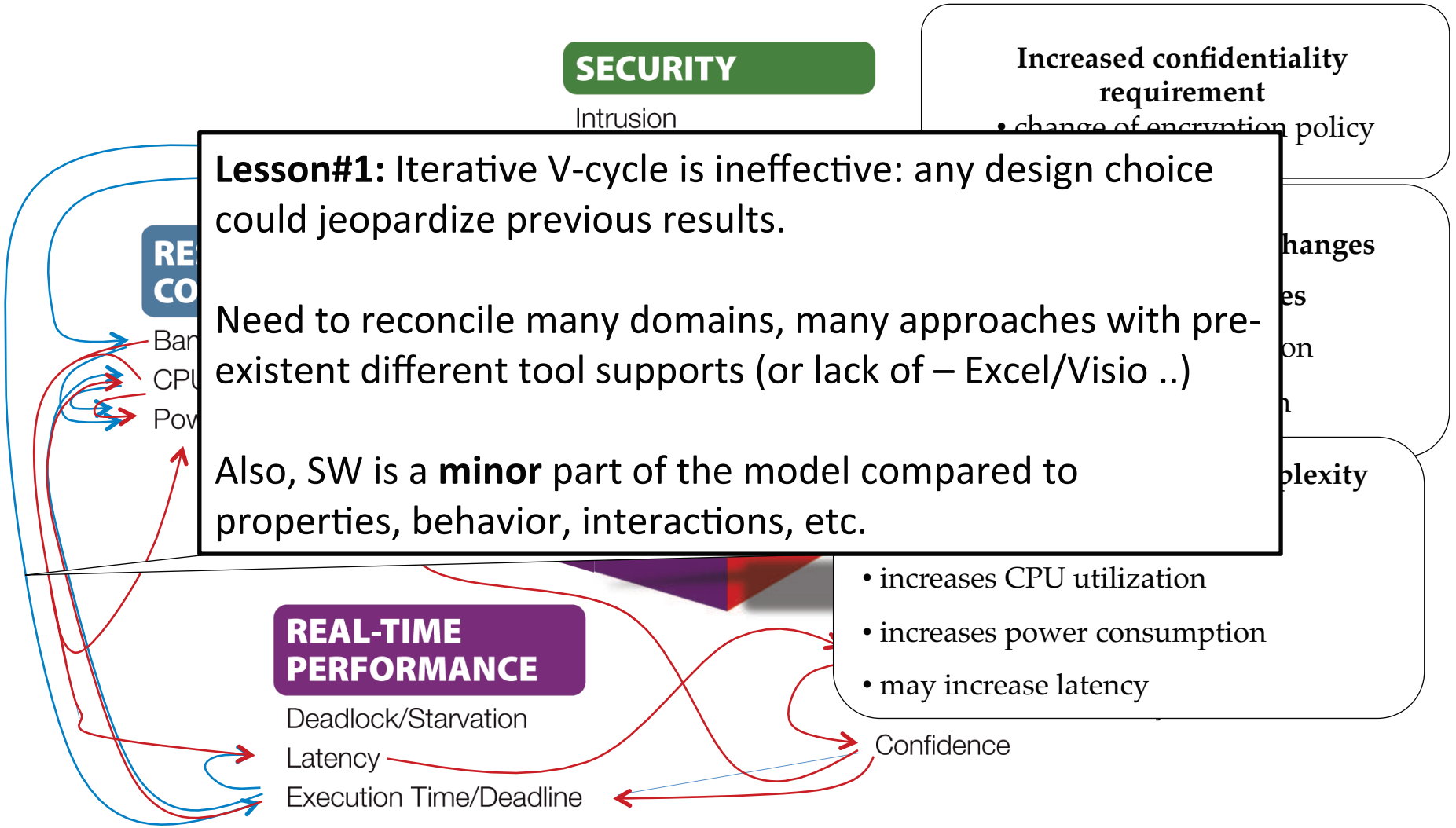


Question: where are the difficulties?
One could simply apply V-cycle, pick tools, and voilà!

- **Supported by AADL**
  - » Scheduling, safety analysis
  - » Model checking, Code generation, …
  - » Single architectural model



Why is analysis in a V-cycle so difficult? (System Engineers 1 – 0 Software Engineers)



Lesson#1: Iterative V-cycle is ineffective: any design choice could jeopardize previous results.

Need to reconcile many domains, many approaches with pre-existent different tool supports (or lack of – Excel/Visio ..)

Also, SW is a **minor** part of the model compared to properties, behavior, interactions, etc.

Why is model-based so difficult?

- **Order of complexity (gratuitous comparison)**
  - » Mathematics: axioms + proof, no interpretation
  - » Electronics, physics, etc.: bound by physics laws, not empirics
  - »
- **M**
  - »
  - »
- **A** **ation”**
  - » E.g. scheduling analysis: true/false
  - » Safety: error rate, stop when below threshold
  - » Also, Analysis part of the GUI space, not the modeling space!

Lesson: Under-specified MDE process incurs 4 out of 7 « wastes » depicted in Lean Management

1. **Waiting:** some analysis are time-consuming, e.g. model checking
2. **Over-processing:** when to trigger analysis?
3. **Over-production:** model too verbose/detailed
4. **Defects:** late discovery of model inaccuracies

⇒ Common root: analysis is a button in the GUI, not part of the model

Need to reflect on analysis process: prog. language can help!

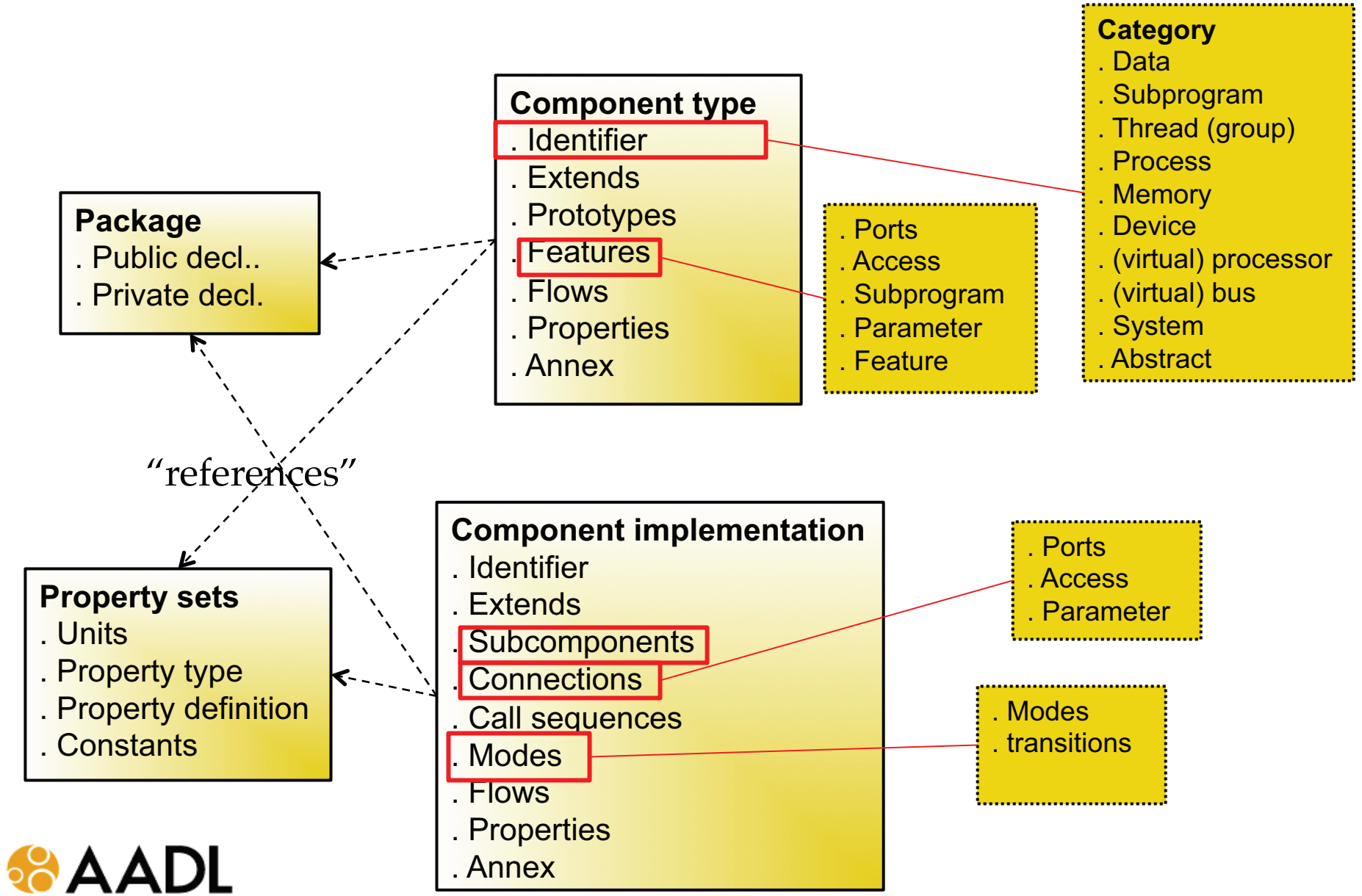
Outline

- Model-Based System/Software Engineering vs. the real world
- **AADL, an overview**
- MBSE as an extension to programming in the large

AADL: Architecture Analysis & Design Language

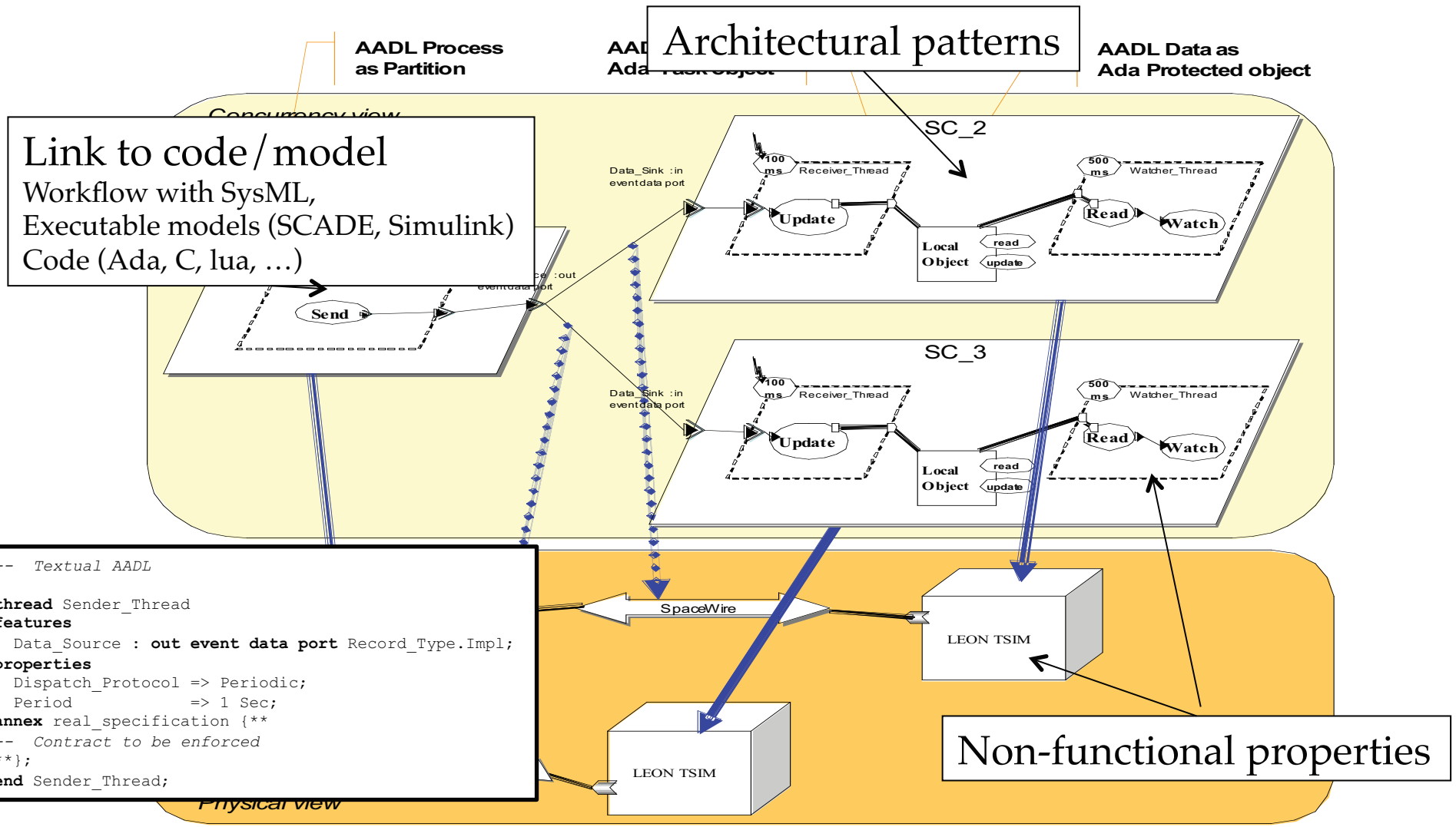
- **International standard promoted by SAE International, AS-2C committee, released as AS-5506A**
- **Version 1.0 published in 2004, version 2 in 2009**
  - Committee driven by inputs from the avionics and space industry
  - Academics drive analysis capability, to ensure they match with modeling patterns
- **http://aadl.info list all resources around AADL**
  - Public wiki with lot of resources: https://wiki.sei.cmu.edu/aadl/index.php/Main_Page
  - Include link to most research activities around AADL
- **AADL is dedicated to real-time embedded domain**
  - Modeling software and hardware resources for V&V
  - Extension & refinements concept to iterate down to generation
- **Different representations**
  - Graphical: high-level view of the system
  - Textual: to view all details
  - XML: to ease processing by 3rd party tool
- **Some interactions with SysML (higher-level design)**

AADL model elements



AADL in one slide (!)

Architecture helps you focusing on the actual system



AADL Extensions

- **AADL is meant to be extensible**
- **New property sets for specific concerns: e.g. ARINC653**
- **Additional language to extend semantics**
  - Behavioral specifications: AADL-BA
  - Error modeling, propagation in a system: AADL-EMV2
  - Constraints on model (on going)
    - Algebraic specifications for contracts, patterns, …
  - Requirement engineering (on going)
- **Each extension has to remain compatible with core**
  - Can be safely ignored if not relevant for a particular objective

Some examples of AADL tool support

- **AADL as a backbone, federating multiple activities**
  - analysis through generation of intermediate models + external tools
- **Common tool IDE: OSATE2 from SEI (FLOSS)**
  - AADL core (SEI) + Behavioral (TPT) + Error (SEI) annexes
- **Non exhaustive list of tools, European-centric (see http://www.aadl.info)**
  - 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, FHA
  - Simulation: ADeS, Marzhin
  - Energy consumption of SoC: OpenPeople project
  - Code generation: SystemC, C, Ada, RTSJ, Lustre
  - WCET analysis: mapping to Bound-T

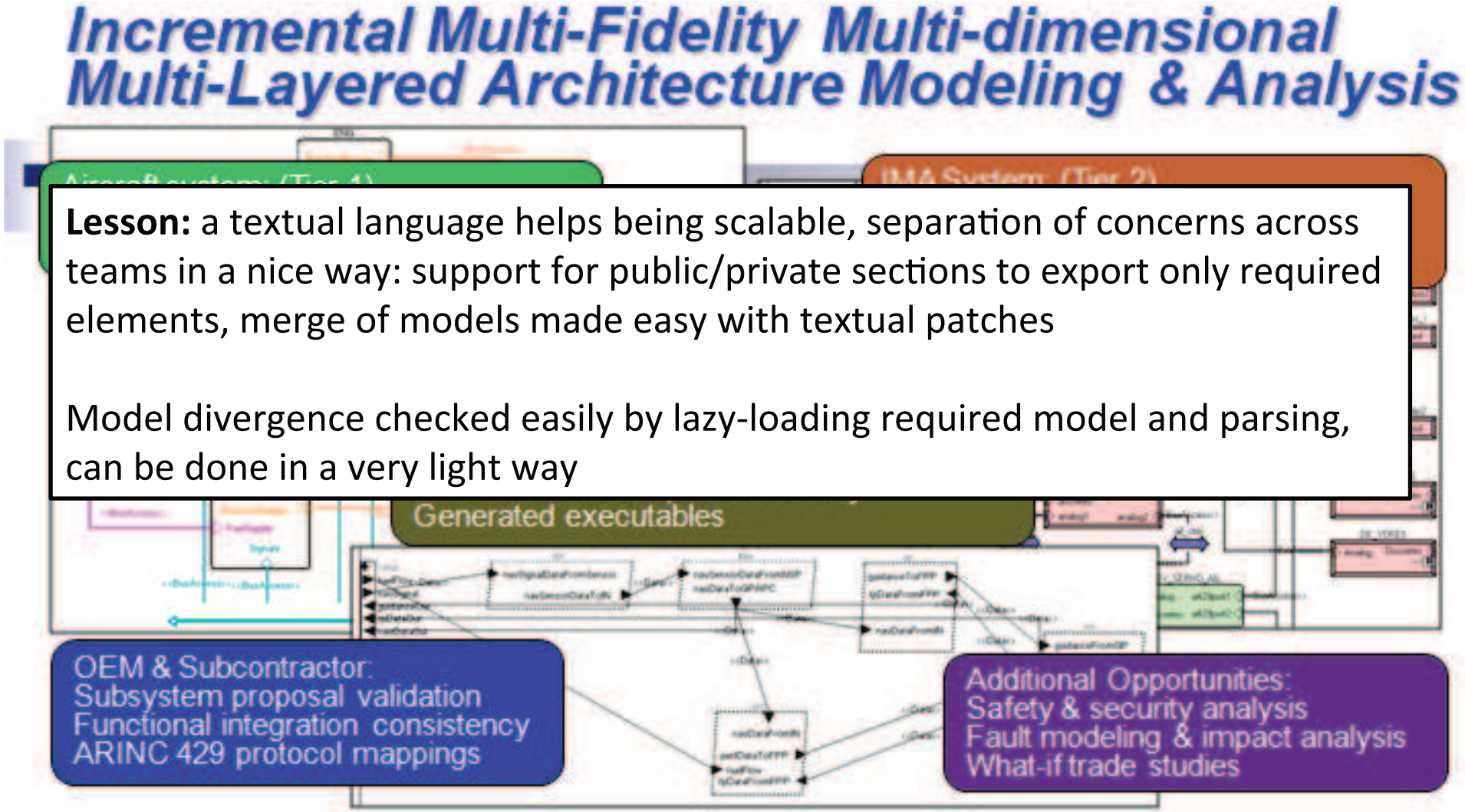
Outline

- Model-Based System/Software Engineering vs. the real world
- AADL, an overview
- **MBSE as an extension to programming in the large**

Moving back to programming language

- **AADL has a concrete syntax**
  - Concrete means also rock-solid to build foundation
- **Scalable: AADL package system close to Ada one**
  - Potential for modular processing
  - Optimizations in representation/processing of the AST
    - OSATE2: EMF, issues with object ids and cache
    - Ocarina: GNAT-like tree: faster, leaner
- **Text also means potential for detailed syntactic constructs**
  - Liskov principle, multiple bindings, formal specs, etc
  - Cannot be (easily) represented graphically !

Example#1: SAVI http://www.avsi.aero



Lesson: a textual language helps being scalable, separation of concerns across teams in a nice way: support for public/private sections to export only required elements, merge of models made easy with textual patches

Model divergence checked easily by lazy-loading required model and parsing, can be done in a very light way

Use of AADL to cover a whole modeling cycle, focusing on validation of high level budgets (mass, energy), interface consistencies, etc.
Modeling teams scattered across multiple teams and companies

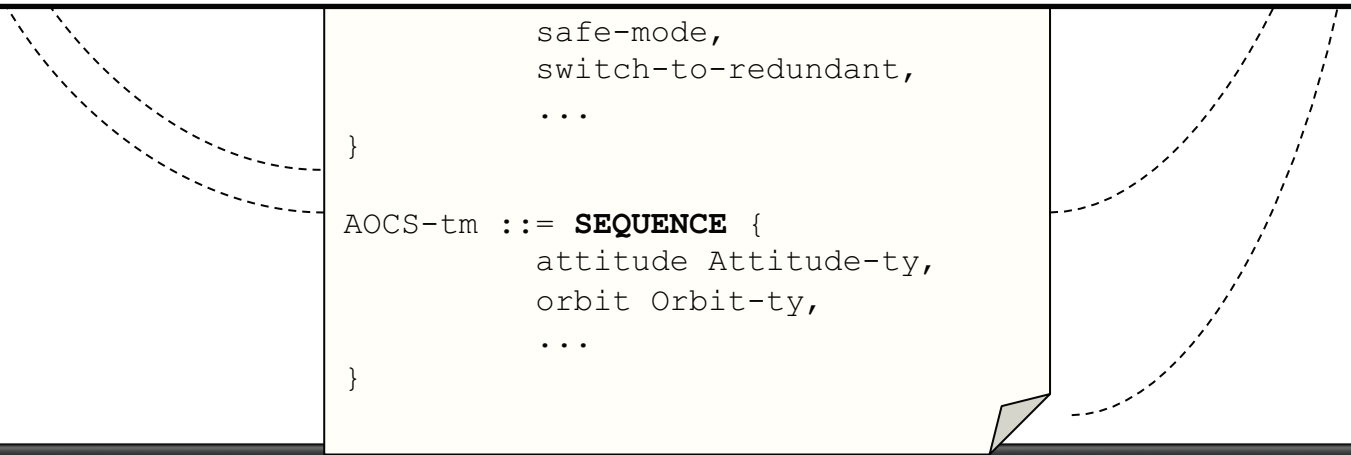
Example#2: TASTE http://assert-project.net/-TASTE-

- **Code generation and analysis for Space-critical systems**
  - » Subset of AADL as input language + model transformations

Lesson: a textual language, free from meta-model management issues is a must to avoid maintenance issues.
TASTE is 7 years old (!), each layer evolves independently, coordinated by an orchestrator

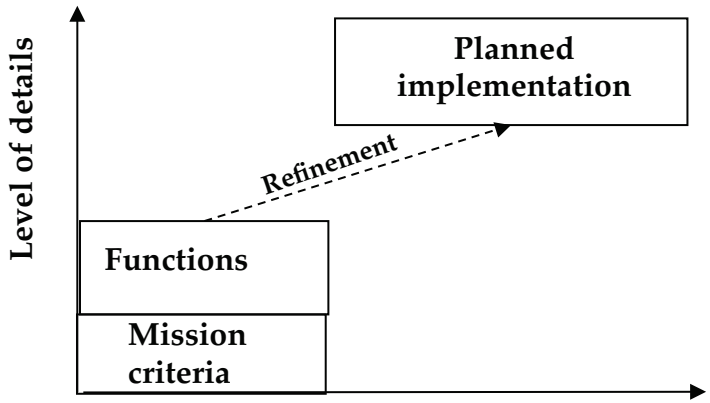
⇒ Each tool either reuses one existing parser (from Ocarina or ANTLR);
⇒ Or simple regexp to find the information it needs.

Simply follow Unix philosophy to address a complex transformation issue

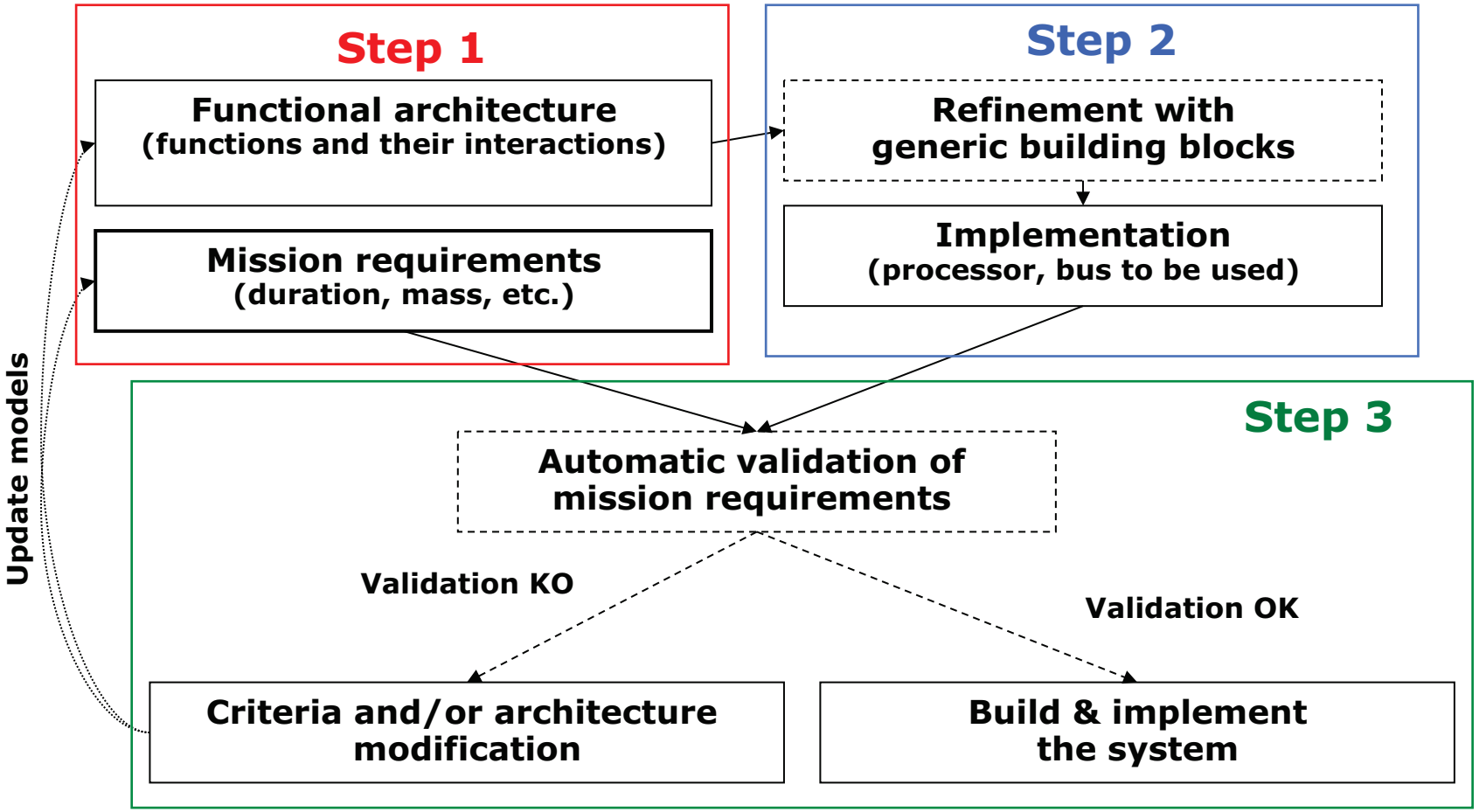


Example#3: ARAM (joint project with ESA, 2011)

- **Based on current practice for space projects at ESA**
- **Define mission criteria**
  - Max weight, orbit position, duration, etc.
- **Specify functional aspects**
  - What will be provided by the platform
  - Specify requirements & constraints
- **Refine the architecture**
  - Replace functions by implementation
  - Reuse existing components
- **Validate planned implementation**
  - Implementation properties vs. Function requirements
  - Automate system integration verification



ARAM Proposed approach, cont'd



System exploration, design, integration

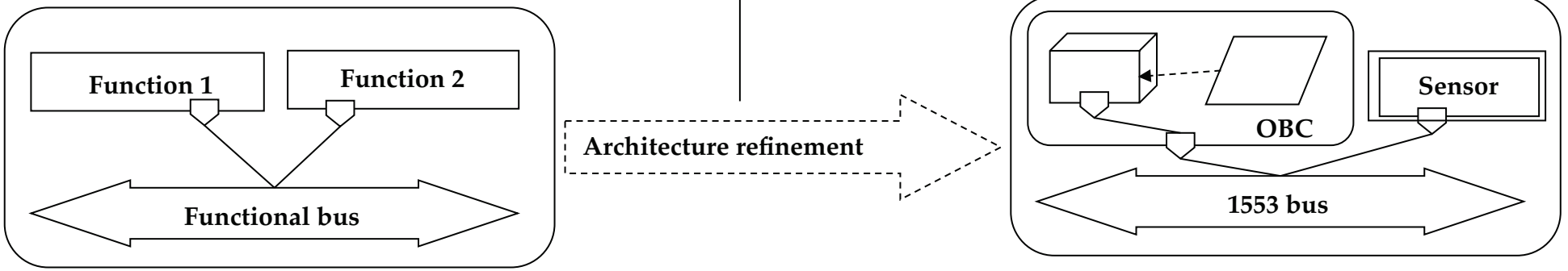
```
abstract  function1
features
   ba : requires bus access
        genericbus;
end function1;
```

```
system implementation mission.i
subcomponents
   f1 : abstract function1;
   f2 : abstract function2;
   b  : bus genericbus;
connections
   bus access f1.ba -> b;
   bus access f2.ba -> b;
annex Constraints {**
 -- list of contracts to be met
**};
end mission.i;
```

```
system obc
features
   ba : requires bus
        access bus1553;
end obc;
```

```
device sensor
features
   ba : requires bus
        access bus1553;
end sensor;
```

```
system implementation mission.planned
       extends mission.i
subcomponents
   f1  : refined to system obc;
   f2  : refined to device sensor;
   bus : refined to bus1553;
-- contracts inherited from mission.i
end mission.i;
```



Contract example

```
-- gaia::functions
abstract fpa_data_get
features
  dataout     : out data port Data_Types::fpa_data;
  ctrlout     : out data port Data_Types::fpa_ctrl;
end fpa_data_get;
-- blocks
device FPA
features
  dataout : out data port Data_Types::FPA_data;
  ctrlout : out data port Data_Types::FPA_ctrl;
properties
  ARAM_Properties::Realizes =>
        (classifier (GAIA::Functions::FPA_data_get));
end FPA;
-- gaia::validation
system implementation Gaia.Validation
subcomponents
  Functional : system GAIA::Functions::Gaia.Functional;
  Impl       : system GAIA::Implementations::Gaia.First_Architecture;
properties
  ARAM_Properties::Actual_Function_Binding =>
    (reference (Functional.get1)) applies to Impl.fpa1.datapart;
```

« Interface » of a function

One implementation

Function/implementation mapping

Function coverage

Lesson: use a common language to model the architecture, **shared** by system engineer and software/hardware engineers

=> Each « role » can model its facet of the model
Syntax and semantics of AADL to bind them all, (like a programming language !)
External model bound to architecture (SysML, Simulink, DOORS, …) for detailed info

=> Each level is attached its own set of verification (constraints, checks, computation, ..) and associated evaluation tool
Refined entities may « inherit » constraints from parent (à-la Liskov)

⇒ Verification rules using DSL evaluate specific architecture patterns,
  ⇒ "if A is connected to B, then the bandwidth of the bus used is less than .."
  ⇒ Part of SAE AS2-C standardization effort

⇒ Nice side-effect: can be used to enforce requirements, subsets, contracts
  ⇒ Used for ARINC, MILS, Ravenscar architectural styles using Ocarina

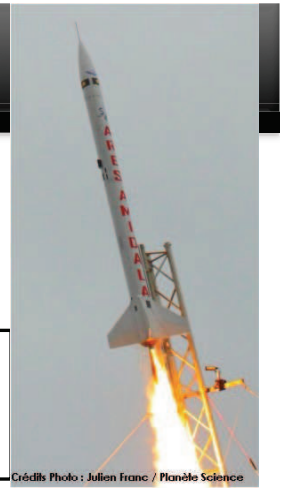
AADL Constraint Language

- **Work in progress as part of SAE AS2-C committee work**
  - Defines accessors and computation rules on model elements
- **E.g. AADLv2 and ARINC653 annex support IMA concepts**
  - Needs to constraint models to respect some invariants

```
theorem scheduling_major_frame --  Check configuration of partition scheduling
    foreach cpu in processor_set do
    check ((float (property (cpu, "ARINC653::Module_Major_Frame")) =
           sum (property (cpu, "ARINC653::Partition_Slots"))));
end scheduling_major_frame;


system IMA_System extends AADL_System - implementation/extension must respect profile
annex real_specification {**
    theorem check_IMA
       foreach s in local_set do
         requires (check_IMA_profile); -- logical conjuction of theorems
end check_IMA; **};
end IMA_System;
```

Example#4: PERSEUS supersonic rocket



> Analysis of rocket kinematics performance

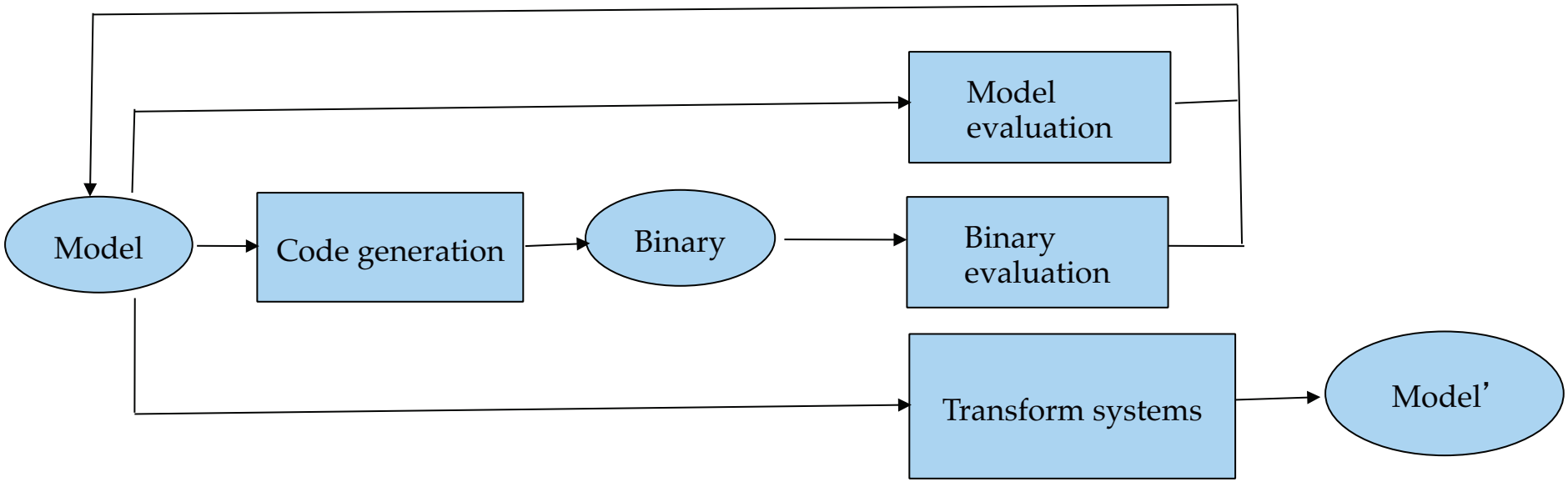
System-level analysis done by combining atomic computations
Each defined separately

Lesson# 2: verification should be driven by domain expert
Expert knows what to compute, the dependencies between parameters
Architects will attach analysis rule to model entities they apply to
⇒ use of AADL annex subclause mechanism

Lesson# 2bis: notion of ordering of rules (*makefile-like*)
Some properties deduced from analysis, reused in another analysis
⇒ Resolution in a compiler AST
⇒ Need also caching and "semantic timestamping"

Example#5: Optimization model/code

- **Combine code generation, scheduling, analysis**
- **Three level of evaluations, combined**
  - Binary: precise evaluation, e.g. memory footprint, WCET
  - Model: check constraints, e.g. requirements or higher-level checks
  - Operation: evaluate the benefits of one modification
  - Under supervision of analysis, scheduling in this context
- **Integrated in Ocarina (O. Gilles PhD)**



Wrap-up (System Engineers 1 – 1 Software Engineers)

- **Equating Model-Based Analysis and Compiling is appealing**
  - Text-based allows for optimization and more precise semantics
    - Fast evaluation for static/simple contracts, proof for complex one (BLESS)
    - Integration of IEEE PSL (dynamic traces) for observers
  - Links with analysis tools made easy
- **Integrating analysis contract to models helps solving**
  - Waiting, Over-processing, Over-production, Defects
  - A compiler/makefile-like approach would optimize analysis effort
    - Run only when required (i.e. model changed “significantly”)
- **Integrating contracts as model elements, and analysis as compilation steps allow for better usage of designer time, and split: analysis designer vs. system designer**