# Oqarina – Mechanization of the AADL Architectural Description Language

Jerome Hugues

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

# BLUF

DSML (Domain-Specific Modeling Languages) required in a model-based context

Graphical and/or textual syntax, complex semantics, multiple analysis capabilities

Common core concepts and typical solutions

- Syntax rules -> BNF
- Typing rules -> meta-model + OCL
- Dynamic behavior -> timed/stochastic transition systems, .. per observed property

This paper: mechanization of AADL, the Architecture and Analysis Design Language

- A large DSML with a rich semantics, many existing tools
- Focus on static semantics
- Two use-cases: user-defined predicates, static scheduling analysis + proofs

# Before You Even Write a Line of Code…

AADL allows you to design the entire system and see where integration problems may occur. Then you can change the design of the system to eliminate those errors.

Being able to perform a virtual integration of the software, hardware, and system is the key to identifying problems early – and changing the design to ensure those problems will not occur.


## About AADL

- SAE Avionics AADL standard adopted in 2004
- Focused on embedded software system modeling, analysis, and generation
- Strongly typed language with well-defined semantics
- Used for critical systems in domains such as avionics, aerospace, medical, nuclear, automotive, and robotics

# AADL Standard Suite (AS-5506 series)

AADL language standard [v1 2004, …  v2.3 2022]

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

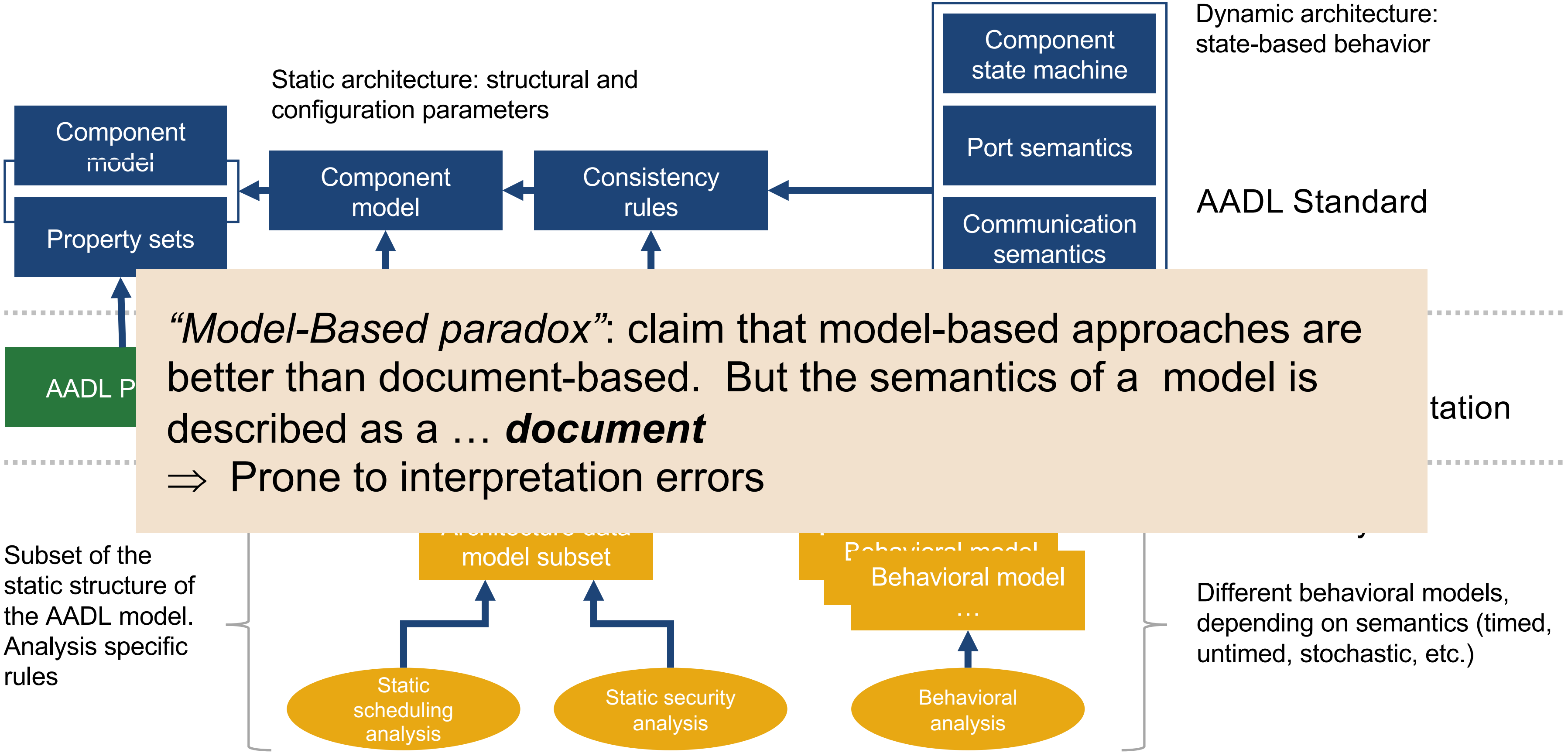
AADL is defined as a language, with a BNF + validity rules

- Implementation choices : meta-model and rule encoding

**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 Layers




*"Model-Based paradox"*: claim that model-based approaches are better than document-based. But the semantics of a model is described as a … ***document***

⇒ Prone to interpretation errors

# AADL Mechanization in Coq

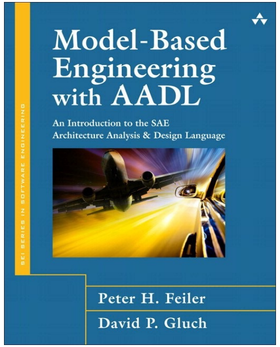
**Research question:** provide unambiguous formal semantics for AADL

- Reference for other tools
- Improved standard by eliminating corner cases

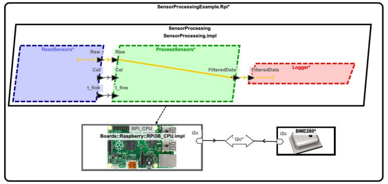
**Solution:** mechanize the semantics of AADL using the Coq Interactive Theorem Prover (ITP)

- Static and dynamic semantics, property sets

Oqarina released as software artefact: github.com/Oqarina under the BSD (SEI) license.




```
Inductive component :=
| Component : identifier → (*  classifier *)
              ComponentCategory → (*  category *)
              identifier → (*  classifier *)
              list feature → (*  features *)
              list component → (*  subcomponents *)
              list property_value → (*  properties *)
              list connection →
              component
  with feature :=
    | Feature : identifier → (*  its unique identifier *)
                DirectionType →
                FeatureCategory → (*  *)
                component → (*  corresponding component instance *)
                list property_value → (*  properties *)
                feature
  with connection :=
  | Connection : identifier →
                 list identifier → (*  path to the source feature *)
                 list identifier → (*  path to the destination feature *)
                 connection
```




SAFIR delivers formal semantics of AADL as Coq types, theorems, and operational semantics.

# From AADL to Coq – Step #1: encoding the grammar

Coq inductive types provide the foundation to encode an AST as a Coq type

```
<category> implementation foo.i [extends <bar>.i]
subcomponents
  -- internal elements
connections
  -- from external interface to
  -- internal subcomponents
properties
  -- list of properties
end foo.i;
```

```
Inductive component :=
| Component : identifier →
    ComponentCategory → (* category *)
    fq_name →
    list feature →
    list component →
    list property_association → compone
(* .. *)
```

Coq typing rules restricts the construction of model elements, e.g. components

# From AADL to Coq – Step #2: Notations

Using the previous terms is not user-friendly

```
Example A_Component := Component
(Id "a_component")
( abstract)
(FQN [Id "pack1" ] (Id "foo_classifier") None) nil nil nil nil.
```

Solution: Coq notations, i.e. a DSML embedded in Coq

```
abstract a_component
features
  a_feature : in event port;
properties
  none;
end a_component;
```

```
Example A_Component_2 :=
abstract: "a_component" → | "pack1::foo_classifier"
features: [ feature: in_event "a_feature” ]
subcomponents: nil
connections: nil
properties: nil
```

# From AADL to Coq – Step #3: legality rules

Legality rules define the correctness of some syntactic statements,

e.g. well-formedness of an AADL component, as a proposition

```
Definition Well_Formed_Component (c : component) : Prop :=
     Well_Formed_Component_Id (c) /\
     Well_Formed_Component_Classifier (c) /\
     Well_Formed_Component_Features (c) /\
     Rule_4_5_N1 (c).
```

A decidable proposition (in `Prop`) denotes a statement that can be proved as either true or false.

(so far) implemented rules are decidable => they can be implemented as Boolean-returning functions

Note: some (minor) reformulations in the standard required to remove ambiguities in order of evaluation for typing rules

# Oqarina

https://github.com/Oqarina/oqarina

- Import model from AADL toolsets: **AADL JSON Parser** →
- AADL meta-model as Coq types: **AADL Instance Coq data type** →
- Is the model "correct" with regards to static semantics rules (e.g., types, static composition): **Well-formedness rules** →
- Behavior of component categories, dynamic composition rules: **Operational semantics**

Coq data types ⇔ AADL meta-model, typing rules, support for building a model

Well-formedness rules ⇔ AADL legality/consistency rules (i.e., model validity)

Operational semantics ⇔ how to "execute" a model (e.g., proof, model checking, simulation)

Features:

- User-defined propositions, Resolute
- mono-core scheduling analysis using the PROSA library
- simulation of an AADL model by mapping to the DEVS formalism *(not discussed today)*

# Oqarina case study #1: Resolute

Resolute is a DSML for reasoning on AADL models, developed by Collins

- First order logic, iteration over component hierarchy, .. for static verification
- Accessors: `is_thread`, `has_property`, `subcomponents`, …

Can be directly embedded in Coq as a library of terms

```
Definition Thread_Has_Valid_Scheduling_Parameters (c : component) :=
     is_thread c ∧
     has_property c Dispatch_Protocol_Name ∧ has_property c Period_Name ∧
     has_property c Compute_Execution_Time_Name.

Definition System_Has_Valid_Scheduling_Parameters (r: component) :=
     All Thread_Has_Valid_Scheduling_Parameters (thread_set r).
```

Coq interpreter used to either compute or prove properties on an AADL model

=> Decidability turns most proof to a mere “trivial” statement.

# Oqarina case study #2: PROSA

Schedulability is one facet of the correctness of a CPS

PROSA supports abstract Response-Time Analysis in Coq

- Data structure lemmas to check schedulability
- Axioms on the system (mono-core, fixed priority) <u>not visible</u>

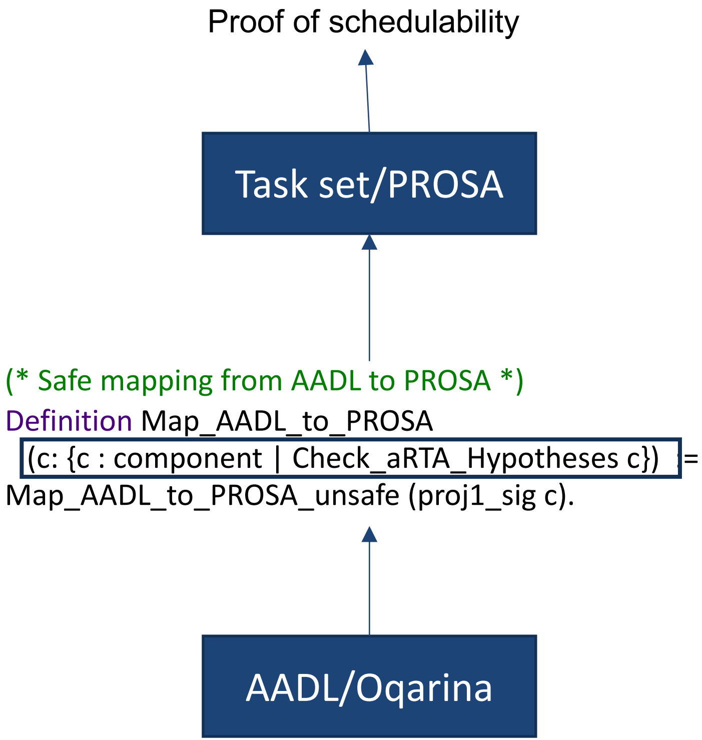
PROSA axioms are decidable properties of AADL models

- Expressed using Resolute

Mapping from AADL to PROSA taskset definition

- translation of concepts (task -> job, priority, WCET, …), guarded by a proof the AADL model is correct

Proof of schedulability using PROSA lemmas



# Conclusion

Mechanizing a DSML in Coq is a feasible task

- Demonstrated static semantics + some verification capabilities
- Approx. 10K SLOCS -- https://github.com/Oqarina/

Dynamic semantics underway

- Defining operational semantics of AADL (see ISOLA'22 paper)
- Translation in Coq and orchestration using the DEVS formalism

$\Rightarrow$ A mechanization of a DSML **+** a proof the DSML semantics is sound

Future work to cover other aspects of AADL: error modeling, flow analysis