AADLv2 as a Domain-Specific Extension to SysMLv2

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.

DM22-0725

An AADL Extension to SysML Is Achievable

SysML's supports systems engineering practice through modeling and model processing.

SAE AADL was started at the request of U.S. Army to address complexity in integrating avionics subsystems through a modeling language and a large class of analysis capabilities.

Recurring concern: Do we need two separate languages?

Working with both impacts cost, model update cycles, etc.

A way forward: the best of both worlds

- SysMLv2 provides a foundation to host the AADL language.
- SysMLv2 open API could bring AADL-native V&V capabilities to the SysML world.

Why Look at SysMLv2 from an AADL Perspective?

AADLv2 defines

- Concepts for representing an architecture: namespace, hierarchy, connection, etc.
- Component categories that specialize these concepts for describing software-intensive systems
- Properties that are typed attributes for configuring component types and instances

→ Notional alignment with SysMLv2 concepts of parts, ports, and attributes

Some AADLv2 extensions are also native in SysMLv2

- SysMLv2 analyses/verification cases and requirements are similar to ALISA (SEI)/Resolute (Collins)
- SysMLv2 behavior has similarities with AADL/BA, SysML1 RAAML with AADL/EMV2 annex

SysMLv2 also has extension mechanisms: new keywords, annex-like constructs similar to AADL

→ How far can we go into specifying AADL as a SysMLv2 library?

AADLv2 Native / AADLv2 SysMLv2 Side By Side

Using SysMLv2 type systems to define AADL type hierarchy

```
package AADLv2_Components
public

    /* Component Type -- AADLv2 */
    thread Thread_1 end Thread_1;
    process Process_1 end Process_1;

    /* Component Implementation */
    process implementation Process_1.impl
    subcomponents
       A_Thread_1 : thread Thread_1;
    end Process_1.impl;

end AADLv2_Components;
```

AADLv2

```
package 'AADLv2 Components' {
    import AADL::* ;

    /* Component Type -- AADLv2 */
    part def Thread_1 :> Thread, Type {}
    part def Process_1 :> Process {}

    /* Component Implementation */
    part def Process_1_impl :> Process_1 {
       part A_Thread_1 : Thread_1;
    }

    /* Instance-like approach */
    part def Process_2_impl :> Process {
       part A_Thread : Thread;
    }
}
```

SysMLv2 + AADLv2 library

AADL Process Category in SysMLv2: Semantics Is an Explicit Artifact

```
part def Process specializes Component {
   attribute redefines category = Component_Category::Process;
   assert constraint {
     checkProcessPorts (portsOnPart) &&
     checkProcessParts (subparts) }}

/* Validity of ports of a process */
constraint checkProcessPorts(p : Port[0..*]) {
   p->forAll { in x : Port ; checkProcessPort(x)}}

constraint checkProcessPort(p : Port) : Boolean[1] {
   p hastype InPort }

/* Validity of parts of a process */
constraint checkProcessParts(p : Part[0..*]) {
   p->forAll { in x : Part ; checkProcessPart(x)}}

constraint checkProcessPart(p : Part) : Boolean[1] {
   p hastype Thread }
```

A Process has a specified category
+ constraints on its parts and ports

Template for constraints on ports
iteration on all ports
+ per-port constraints

Template for constraints on parts
iteration on all parts
+ per-part constraints

Modeling Properties and Property Sets

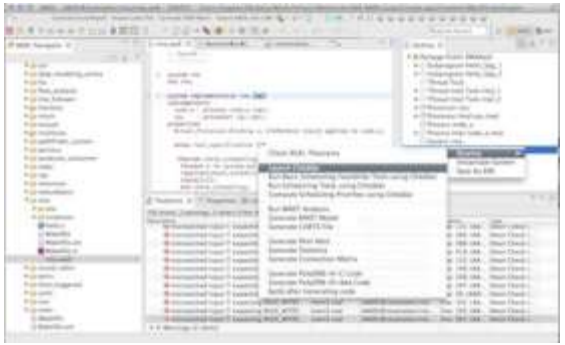
Property sets as configuration sets, similar to AADLv2 concepts

```
/* Configuration set */
abstract part def Thread_Scheduling_Properties {
        attribute Dispatch_Protocol : Supported_Dispatch_Protocol;
        attribute Priority : Integer;

        /* These properties only applies to Threads. */
        assert constraint { self hastype Thread }
}

part def A_Thread :> Thread, Thread_Scheduling_Properties {
        attribute redefines Priority = 42;
        attribute redefines Dispatch_Protocol =
                Supported_Dispatch_Protocol::Periodic;
}
```

About Tool Support

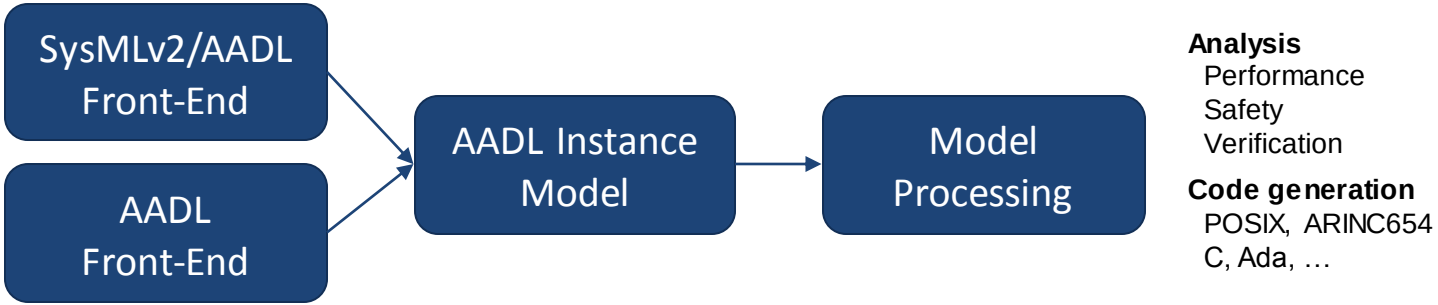


OSATE is the reference implementation of AADL.

- Eclipse/EMF based
- 3rd-party tools by SEI, Collins, Adventium, GE, ..

Rely on an internal representation of an AADL model “Instance model”

➡ Possibility to adapt them to SysMLv2 AADL library



Conclusion

SysML and AADL synergies

- Systems engineering applies to safety-critical systems
- Similar language concepts, different tool ecosystems

SysMLv2 language capabilities can support AADL concepts.

One language ..

- Specialization for safety-critical embedded systems
- Joint tooling thanks to SysMLv2 API for interoperability

Opportunities as SysMLv2 finalizes

- Finalize library and associated technical report
- Reach agreement among practitioners and stakeholder for publication as a standard
- Tool updates as a longer-term objective