
Julien Delange, Laurent Pautet, TelecomParisTech,
Jérôme Hugues, ISAE/DMIA
Dionisio de Niz, SEI/CMU
About the SAE AADL

- **AADL model**: A hierarchy from top-most (system) to top-down (subprograms, ...)
- **AADL components**: 
  - **Component definition**: model of a software or hardware element, notion of type/interface, one or several implementations organized in package. A component implementation may have subcomponents.
  - **Component interactions**: features (part of the interface) + connections (access to data, to subprograms, ports, ...)
  - **Component properties**: valued attributes to model non-functional property (priority, WCET, memory consumption, ...)
- AADLv2 defines **both** textual and graphical representations
- UML/MARTE defines guidelines for modeling AADL
PACKAGE radar
PUBLIC

PROCESS processing
-- ...
END processing;
DEVICE antenna
-- ...
END antenna;
END RADAR;

AADLv2 Radar example
AADL and subprograms

- Default AADLv2 properties / AADL runtime allows one to bind user code to AADL model
  - This code is then executed e.g. when a thread is dispatched
  ```
  subprogram Receiver_Spg
  features
    receiver_out : out parameter Target_Distance;
    receiver_in : in parameter Target_Distance;
  properties
    Source_Language => Ada95; -- defined in AADL_Project
    Source_Name => "radar.receiver";
  end Receiver_Spg;
  ```
- Nothing prevents inclusion of models as “source code”, e.g. SDL, Scade, Simulink or Esterel
- **Issue:** how to perform this consistently?
AADL and other modeling notations

• AADL is an interesting framework to model architectures
  ➢ Capture key aspects of design: hardware/software
  ➢ Expression of some non functional properties: priority, resource consumption, latency, jitter, …
  ➢ Enables: scheduling analysis, resource dimensioning, behavior analysis, mapping for formal methods, fault analysis, …

• Functional modeling notations (e.g. Simulink, SCADE, ..) describes precisely how the system should behave
  ➢ Provides a high-level behavioral/computational view
  ➢ Needs to be mapped onto hardware/software elements

• Natural complement to build systems with models
  ➢ Without hand-written code
"Zero coding" paradigm

- Code generation from models is now a reality
  - Proposed by many tools
- Functional models
  - kcg: SCADE’s certified code generation
  - Real-Time Workshop: Simulink’s code generation
- Architectural models
  - Ocarina: AADL code generator for High-Integrity systems
- Foundations for a “zero coding” approach
  - Model, then integrate code generated from each view
- **Issue**: which integration process?
  - Two approaches, driven by user demand
Application-driven process

- Functions may be defined first, then refined to be bound to an existing architecture
Architecture-driven process

- Reverse option: architecture is defined first, then a skeleton of the functional model is deduced, then implemented

```plaintext
subprogram spg_scade
features
input: in parameter integer {Source_Name => "add_input"};
output: out parameter integer {Source_Name => "add_output"};
properties
  source_name => "inc";
  source_language => Scade;
  source_location => "/path/to/scade-code/";
end spg_scade;
```
How to bind to AADL models?

• In both cases, we rely on standard AADLv2 patterns
  ➢ Source_Language <-> SCADE or Simulink
  ➢ Source_Name <-> SCADE node or Simulink block
  ➢ Source_Location <-> SCADE/Simulink generated code

• Smooth integration of AADL and other functional modeling
  ➢ Providing only required information
  ➢ While remaining 100% automatic
From AADL + X to code

- Ocarina is an AADL-to-code generator
  - See [http://aadl.telecom-paristech.fr](http://aadl.telecom-paristech.fr)
  - Joint work Telecom ParisTech, ENIS, ISAE
- Handles all code integration aspects
  - How to map AADL concepts to source code artefacts (POSIX threads, Ada tasks, mutexes, ...)
  - Handle portability concerns to several platforms, from bare to native
- + some knowledge on how a SCADE or Simulink models is mapped onto C code
  - So that integration is done by the code generator
  - No manual intervention required
- Supports “zero coding” approach
Code generation patterns

- Each functional framework relies on same foundations
  - Synchronous: discrete computation cycles
  - Asynchronous: function calls
- SCADE/Simulink/Esterel: a 3-step process
  - Fetch **in** parameters from AADL subprograms
  - Call the **reaction function** to compute output values
  - Send the output as **out** parameters of the AADL subprogram
- Architectural blocks are mapped onto programming language equivalent constructs
  - Ocarina relies on stringent coding guidelines to meet requirements for High-Integrity systems, validated though test harness by ESA, Thales, SEI, and their partners
Conclusion

- System are heterogeneous, so are models
- AADL clearly separates architecture from functional models
  - Allows reference from the architecture to function blocks
- **Our contribution:** integration of AADL and SCADE or Simulink in two processes to perform full generation of systems
- Advantages
  - “Zero coding” paradigm to ease integration work
  - Quality of code generated for both functions and architecture
  - Opens the path towards qualification/certification of complex embedded systems at model-level