



AADL to build DRE systems, experiments with Ocarina

Jérôme Hugues, ENST



ENST Research topic: Methods for DRE

- Building a DRE is still a complex issue:
 - ⇒ RT-CORBA, DDS are only partial solutions
 - ⇒ Still difficult to analyze (scheduling, dimensioning)
- Goal: propose a methodology, middleware and tools for DRE
 - ⇒ Validation & Verification, configuration, deployment
 - ⇒ Automate the process as much as possible
 - ⇒ Scale up to complex systems
 - ⇒ Ensure reusability (process, code, models, know how, etc)
- Architecture as key enabler to build DRE systems
 - ⇒ Key to system validation, scalability, support for application, ...

How to solve this challenge ?

- **Key challenge: configuration of the middleware**
 - ⇒ The architecture governs both the configuration and deployment
 - ⇒ Needs a (simple) way to express both
 - ⇒ Should not impede late binding decision such as selection of the run-time environment, model analysis tools
 - Difficult to achieve with UML/profiles/meta-models (for now)
- **Pragmatic approach: go back to basics of software architecture**
 - ⇒ Select a modeling language for architecture description
 - ⇒ Need a language to express architecture, with analysis and enough expression power to describe
- **AADL as a vehicle to address (most) issues**

Our approach

- One common design philosophy
 - ⇒ Extensions through design refinements
 - ⇒ Promote late binding decision to promote reuse
- Two complementary technologies
 - ⇒ AADL, Ocarina
 - Design language for HRT systems
 - Connections with code generators and verification tools
 - ⇒ “Schizophrenic middleware”, PolyORB
 - Both a design methods and its supporting implementation
 - Extreme genericity of middleware constructs to support new technologies (RT-CORBA, DDS), fine grained tuning
 - PolyORB: QoS-based middleware
 - “PolyORB-HI”, middleware for HI systems
 - A runtime for the AADL

Architecture Analysis & Design Language

- SAE-AADL, an ADL for distributed real-time embedded (DRE)
 - ⇒ Focuses on high-level components down to software/hardware concerns
- Integration of separately developed components
 - ⇒ Interfaces, refinement of components, assembly
- Provides precise & machine-processable syntax
- Allows (formal) analysis of the properties of the architectures
 - ⇒ Resource allocation, execution time, ...
- Advocates generation of a system from its description
 - ⇒ Mapping to languages (C, Ada, SimuLink, VHDL, ...)
- AADL is a common support for many information

AADL (short) Overview

- AADL Description = set of components
 - ⇒ Component = 1 interface [+ 0 .. N implementations]
 - ⇒ Some components can contain subcomponents
- Components communicate through features, described in the interfaces
 - ⇒ Features are ports, accesses to subcomponents, etc.
 - ⇒ Features are connected using connections
- Components
 - ⇒ Software: Data, Process, Thread, Subprogram
 - ⇒ Execution platform: Memory, Processor, Bus, Device
 - ⇒ System
 - Contain other components
 - Structure the system
- Properties associated with elements: Features, Connections
- Standard properties
 - ⇒ Resource usage
 - ⇒ Behavioural descriptions
 - ⇒ User-defined property sets

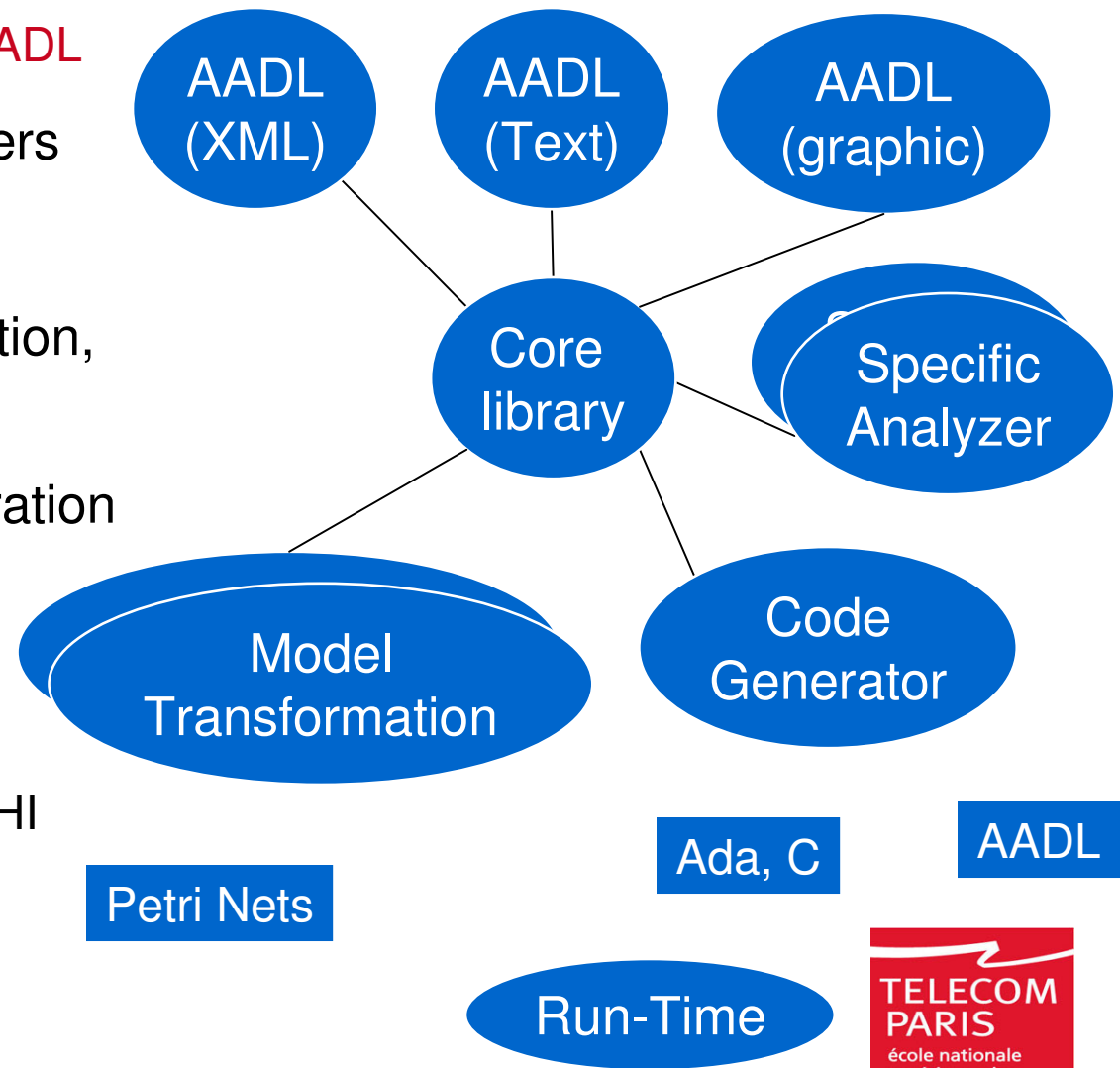
Using AADL

- AADL is a vehicle to express many architectural elements
 - ⇒ Interacting entities + Non-functional properties
- AADL allows for
 - ⇒ Model analysis & code generation
- Current technologies at ENST
 - ⇒ Ocarina: AADL tool set
 - ⇒ PolyORB HI + Ada runtime: middleware and RTOS
 - ⇒ 3rd tools: CPN-AMI (Petri nets), Cheddar (scheduling)
- Partnerships within IST-ASSERT (AADL), AdaCore & ObjectWeb (middleware), Thalès (AADL), ARTIST2, ARTEMIS, SAE (AADL)

Ocarina Tool Suite

- **Library to manipulate AADL**
 - ⇒ Parsers and printers
 - ⇒ Semantic checks
 - ⇒ Model transformation, code generation
 - ⇒ Run-time configuration

- **Ocarina 1.0**
 - ⇒ Code generator
 - Ada/PolyORB
 - Ada/PolyORB-HI
 - ⇒ V&V
 - Schedulability (Cheddar)
 - Petri Nets



HI-middleware for AADL: PolyORB-HI

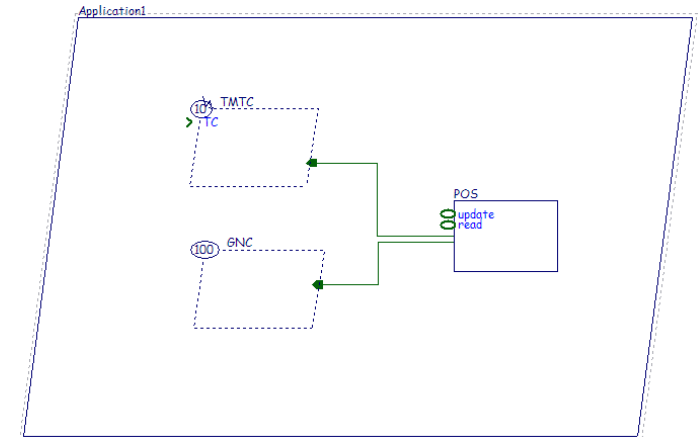
- AADL defines a semantics for a runtime
 - ⇒ Possible to derive code from the architecture
 - ⇒ Containers completed with application-specific code
- Hard-Real Time systems specific constraints
 - ⇒ Ravenscar profile + Ada High-Integrity restrictions
 - ⇒ No allocator (task, memory), no dispatching
- Exploit AADL properties and patterns to generate code
 - ⇒ Runtime entities correspond to some AADL patterns
 - ⇒ Configured through AADL properties (priority, stack, protocol, ...)
- Code generation strategies
 - ⇒ Abiding to HI restrictions, Ada compilation checks, etc.

Process to build applications

- Combine AADL, compilers, model checkers, validation tools
 - ⇒ Automatize as much as possible the analysis of a system
- Fully automatized process
 - 1. Stack and memory dimensioning (tool support by GNAT)
 - 2. Schedulability analysis (Cheddar)
 - 3. Petri Nets to assess system's behavior (Ocarina)
 - 4. Construction of each deployed node + middleware configuration (Ocarina)
 - 5. Source code generation (Ocarina)
 - 6. Compilation, metrics (GNAT)
 - 7. Execution (GNAT for ERC32, LEON2, native platforms)

Building application from AADL & code

- Application sample (control system)
 - ⇒ Periodic, aperiodic, shared variable
- Refinement from high-level view to deployment
 - ⇒ Model + function code
- Down to code generation
 - ⇒ System is schedulable
 - ⇒ Has no deadlock
 - ⇒ Can be generated for PolyORB-HI
 - ⇒ Fully respects all compile-time restrictions
 - ⇒ Then runs on **tsim** (LEON2 simulator)
- Ready for certification & deployment
 - ⇒ System comes with models and code



```
system implementation toy_example.sample_1
subcomponents
  P1 : processor the_processor;
  P2 : processor the_processor;
  GNC : process GNC_Proc;
  TMTC : process TMTC_Proc;
properties
  Actual_Processor_Binding
    => reference P1 applies to GNC;
  Actual_Processor_Binding
    => reference P2 applies to TMTC;
end toy_example.sample_1;
```

Ping example

- Two remote nodes:
 - ⇒ One cyclic task pinging another one, on different nodes
- Based on a simple distribution model,
 - ⇒ Holistic analysis feasible
 - ⇒ Bounded types
 - ⇒ Static configuration from the AADL model
- Distribution expressed as a late binding decision
 - ⇒ One last step of refinement of the system, when allocating threads to process and then processor
 - ⇒ For now, TCP/IP communications, SpaceWire to come

Case Study for AADL

- Taken from ASSERT D3.1.3-1, proposed by EADS
 - ⇒ GNC is a cyclic process, with an activation period P . Its maximal CPU consumption is CPU_GNC . Its deadline is $DLGNC < P$.
 - ⇒ TMTC is an acyclic process, activated on reception of the event TC. Its maximal CPU consumption is CPU_TMTC . Its deadline is $DLTMTC \ll DGNC$ (for instance, $DLGNC = 10 * DLTMTC$).
 - ⇒ POS is a shared variable between GNC and TMTC:
 - GNC reads POS every P seconds. It performs a computation and then updates the value of POS at the end of its computation.
 - TMTC updates the value of POS on reception of TC. The write of TMTC shall always be taken into account. It can overwrite a write of GNC or a previous write of TMTC
 - The write of GNC can be overwritten by TMTC. If TC occurs when GNC is not active, POS can be immediately updated. If TC occurs when GNC is active, the update of POS has to be delayed until the termination of GNC

Conclusion and Ongoing Work

- No “one size fits all” middleware for DRE
 - ⇒ Many configuration points, difficult to use them
- Methodology
 - ⇒ to elaborate customized middleware solutions
 - ⇒ based on highly configurable middleware solution and architecture description language
- PolyORB: two complementary middleware
 - ⇒ QoS: RT-CORBA, DDS for “big” targets
 - ⇒ HI: for “small” targets, with strong constraints (High-Integrity)
- Ocarina: generation for both MW + verification + scheduling
- Ocarina 1.0 (released Jan. 2007), see <http://ocarina.enst.fr>
 - ⇒ Better support for AADL, nightly snapshots for interested users
- Ongoing work in the context of IST-FP6 ASSERT (ESA, ...)