From MDD back to basic: Building DRE systems

Jérôme Hugues, ENST

MONTEREY'06

MDx in software engineering

- Models are everywhere in engineering, and now in software engineering
- MD[A, D, E] aims at easing the construction of systems
  - Enforce careful modeling through clear modeling language/artifacts
  - Support for model transformation, analysis, verification, …
  - Combine many expertise in one process
    - ✓ *Requirement capture, dimensioning, scheduling analysis, verification*
    - ✓ *Tools support required, and becoming mainstream (at least partially)*
- Building a DRE remains a complex issue
  - RT-CORBA, DDS are only partial solutions
  - Difficult to build large systems, to combined app. components
  - Harder to complete system analysis (scheduling, dimensioning)
- Solution (obvious ?): apply MDD to deploy and use middleware
  - How ? What are the pitfalls ? Which reasonable direction ?

Some thoughts on MDx

- MDx aims at more efficiency
  - By manipulating high-level abstractions
  - By leaving tedious and error-prone work to tools & automation
- Models are easier to understand & to build systems than code
- But
  - Model implementations cannot be fully substituted to code
  - Benefit from code generation & model transformation still ahead
- MDx is no need without
  - a well-defined modeling "cookbook"
  - a clear engineering method and design rules
  - a proven engineering process sustaining both
- Efficiency largely relates on toolset capability

MDx & DRE systems

- Two pitfalls
- Abstraction inversion
  - Use something complex to re-implement something simple that already exists
  - Ex: CORBA CCM & DanCE to support simple event propagation
- Chicken & egg dilemma
  - Do you model a system ? Or do you conform to a meta-model ?
  - Of course, meta-model constraints model construction
  - BUT a common issue is that meta-model constrains the system too much
    - ✓ *Semantics of the underlying RTE (RTOS, middleware) should appear wisely*
    - ✓ *And should not prevent model portability or adaptability !*

Need to go back to basic, to build DRE systems on simple patterns

To ease model reusability, portability

=> Focus on the architecture first <=

Families of DRE middleware

- Middleware is software that has to offer more
  - High configurability, performance,
  - Support for more targets, more constraints
- Similarities with typical software engineering
  - Middleware is complex by nature
  - Should be adapted to application domains (avionics), families (safety)
  - Support for semantics (DOC, MOM), standards (RTCORBA, DDS)
  - And configurability (QoS policies, mapping of app. entities onto nodes)
- Middleware support distribution on behalf of application
  - QoS-based: "large" embedded system, RT-CORBA, DDS
  - HI: implementation is semantically constrained for certification
- Middleware is a point of failure of the system, to be avoided
  - Some errors: configuration, deployment, interpretation of the semantics
- Models to address all these complexity issues ?

How to solve this challenge ?

- Two middleware families, one 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
    - ✓ *Difficult to achieve with UML and profiles/meta-models (for now)*
- Goal: propose a methodology, middleware and tools to build DRE
  - Automate validation & verification, configuration, deployment
  - Scale up to complex systems
  - Ensure reusability (process, code, models, know how, etc)
- Pragmatic approach: go back to basics of software architecture
  - Select a modeling language for architecture description
  - Exploit the description of the architecture to ensure system correctness
  - Need a language to express architecture, with analysis and enough expression power to describe
- An Architecture Analysis & Design Language ? You named it, AADL ;)

Our approach

- One common design philosophy to build systems
  - Extensions through design refinements
  - Promote late binding decision to enforce reuse

- Two complementary technologies
- AADL, Ocarina
  - Design language for HRT systems, support design by refinement,
  - Connections with code generators and verification tools
- "Schizophrenic middleware", PolyORB
  - Both a design methods and its supporting implementation
  - Extreme genericity of middleware constructs
  - Enable the rapid prototyping of middleware (RT-CORBA, DDS, dedicated)
    - ✓ *PolyORB: QoS-based middleware*
    - ✓ *"PolyORB-HI", part of IST-ASSERT, middleware for HI systems*

Architecture Analysis & Design Language

- AADL, an ADL to describe distributed real-time embedded systems
  - A standard proposed by SAE *(Society of Automotive Engineers)*
  - Focuses on high-level components down to software/hardware concerns
- Integration of separately developed components
  - Clear interfaces, refinement of components
  - Component assembly
- Provides precise & machine-processable syntax
  - Text and graphical notations, link with UML
- Allows (formal) analysis of the properties of the architectures
  - Resource allocation, execution time, …
- Advocates generation of a system from its description
  - Mapping to programming languages (C, Ada, SimuLink, VHDL, ...)
  - Definition of a run-time

- 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

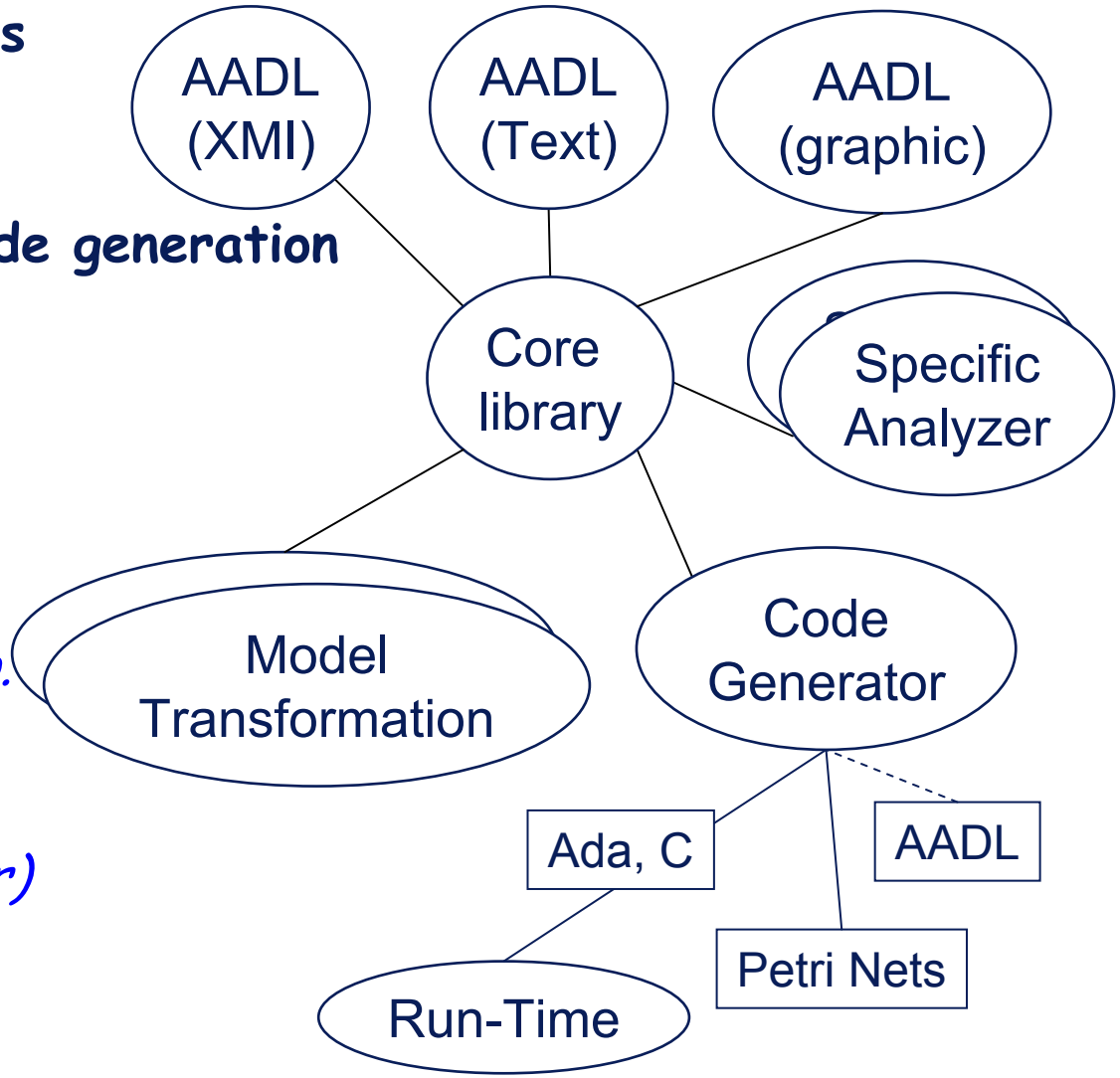
- Components
  - Software
    - ✓ *Data, Process, Thread, Subprogram*
  - Execution platform
    - ✓ *Memory, Processor, Bus, Device*
  - System
    - ✓ *Can contain other components*
    - ✓ *Structure of the architecture*

- Properties associated with elements
  - Components / subcomponents
  - Features
  - Connections
- Standard properties
  - Resource usage
  - Behavioural descriptions, …
- Property sets
  - For user-defined properties

Ocarina Tool Suite

http://ocarina.enst.fr

- Library & tools to manipulate AADL
  - AADL parsers and printers
  - Semantic checks
- Specific operations
  - Model transformation, Code generation
  - Run-time configuration
- Provides
  - Code generators
    - ✓ *Ada/PolyORB*
    - ✓ *Ada/PolyORB-HI*
    - ✓ *Both local and dist. App.*
  - V&V
    - ✓ *Petri Nets*
    - ✓ *Schedulability (Cheddar)*

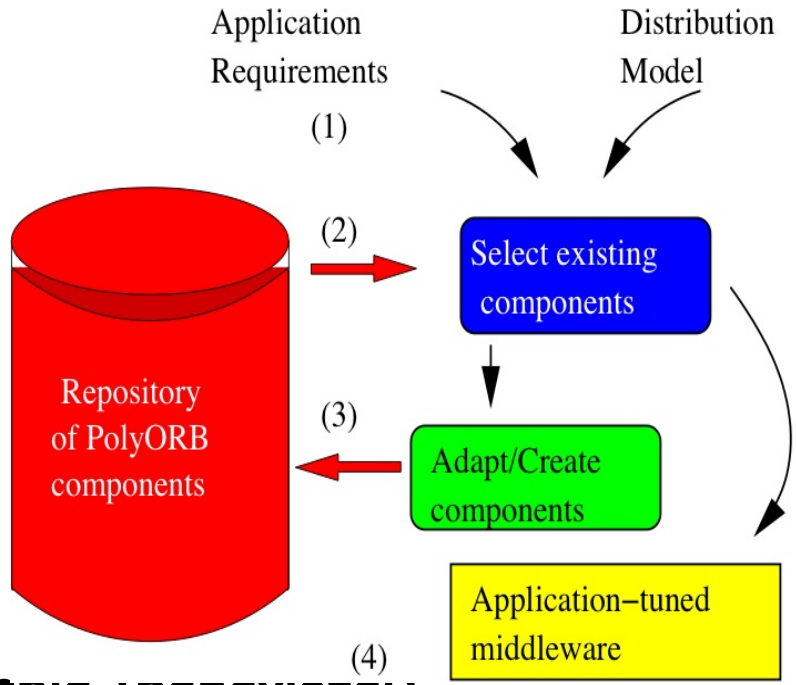


Reusable Framework for DRE systems

- Reorganize middleware functionalities to reduce components coupling
  - like an OS on top of a micro-kernel
- Define generic building blocks to describe middleware: Addressing, Binding, Representation, Protocol, Transport, Activation, Execution
- Let interaction between building blocks be independent from any specific distribution model
  - Common behavioral contract => ease modeling
- Propose one implementation for each generic building block
  - Enable code reuse
  - Tailoring for specific needs whenever required
- Generic services propose a coarse grain parameterization
  - Components refinement, OO techniques, implement new behavior, ..
- Configuration is fine grain customization of blocks
  - Selection of one strategy, one parameter value, ..

QoS-middleware: PolyORB
Configurable, Generic and Verifiable Middleware

- PolyORB is our reference middleware implementation
  - Configurability and extreme genericity
  - Clear design: modeling and formal verification
- Already support many distribution facilities
  - CORBA (RT-, FT-, MIOP), OMG DDS
  - MOM (MOMA), SOAP, Web Apps.
- Each facility enriches MW repository
  - Remote reference, protocol, representation
  - Request demultiplexing, concurrency
  - QoS policies and management
- Components be either specific (API) or reusable (behavioral)



http://polyorb.objectweb.org

- Challenge: configuring the many semantics variation points
  - Tools and models to help configuring the middleware

HI-middleware: PolyORB-HI
Building Distributed Hard-Real Time Systems

- Hard-Real Time systems specific constraints:
  - Ravenscar profile + Ada High-Integrity restrictions
  - No allocator (task, protected object, memory), no dispatching
- Building HI middleware is a pain, building an application a nightmare
  - Careful allocation of all objects (dimensioning)
  - Difficult to select configuration points (typical factories forbidden)
  - Careful analysis of concurrent constructs with certification in mind
  - Object-orientation defeated by HI constraints, must hard-code many things
- Existing prototype for ASSERT: "PolyORB-HI"
  - Compliant with all HI constraints, numerous Ada checks, etc.
- Challenge: suppress tedious and error-prone process of writing apps.
  - Tools and models to express constraints, and then generate code
  - Go through V&V process

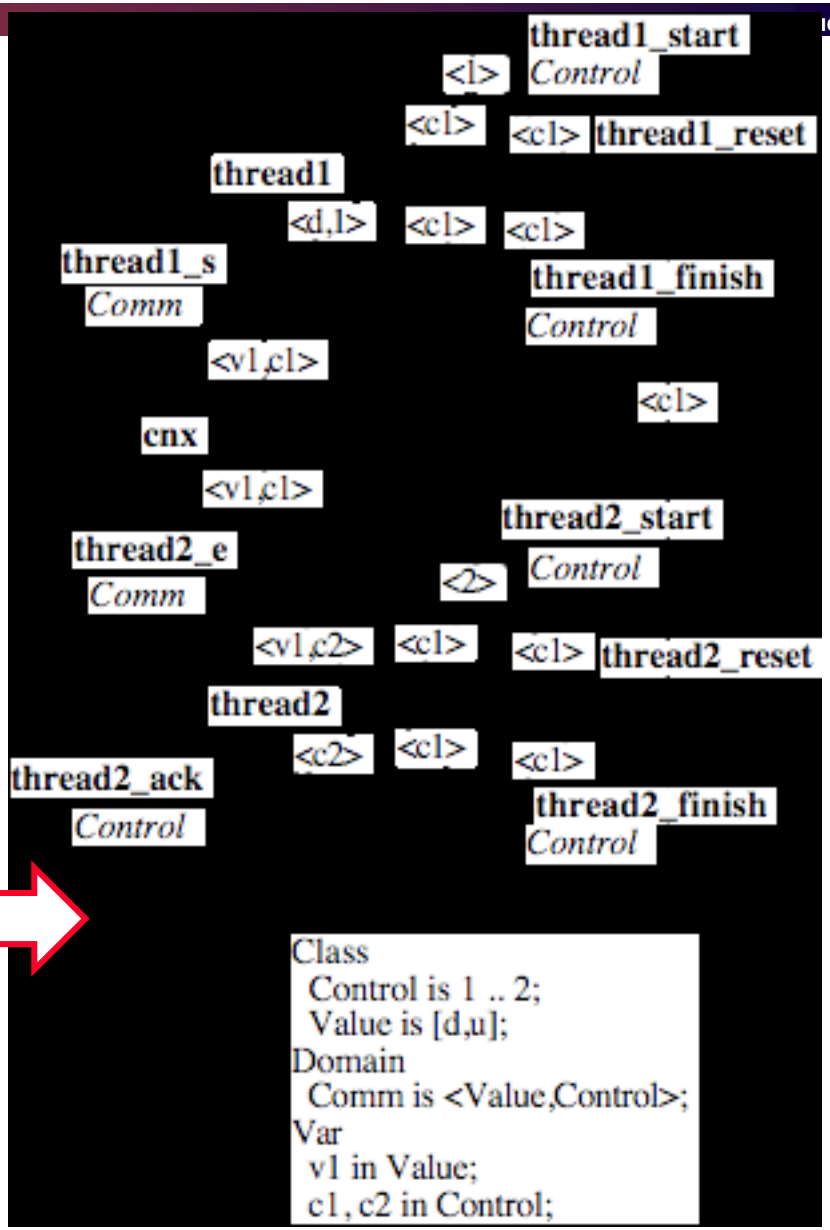
Process to build applications

- Combine AADL, compilers, model checkers, validation tools, run-time environments in one unified process
  - From early design to final implementation
  - Automate as much as possible the analysis of a system in each step
  - Add code generation from AADL to well-defined code patterns
- Integrated process, combining Ocarina and 3rd-party tools
  1. Syntax checking (Ocarina)
  2. Stack and memory dimensioning (GNAT or Cheddar, not yet integrated)
  3. Schedulability analysis (Cheddar)
  4. Petri Nets to assess system's behavior (Ocarina)
  5. Construction of each deployed node + middleware configuration (Ocarina)
  6. Source code generation (Ocarina)
  7. Compilation, metrics (GNAT)
  8. Execution (GNAT for ERC32, LEON2, native platforms)
- Experiments in the IST-ASSERT project, first results promising

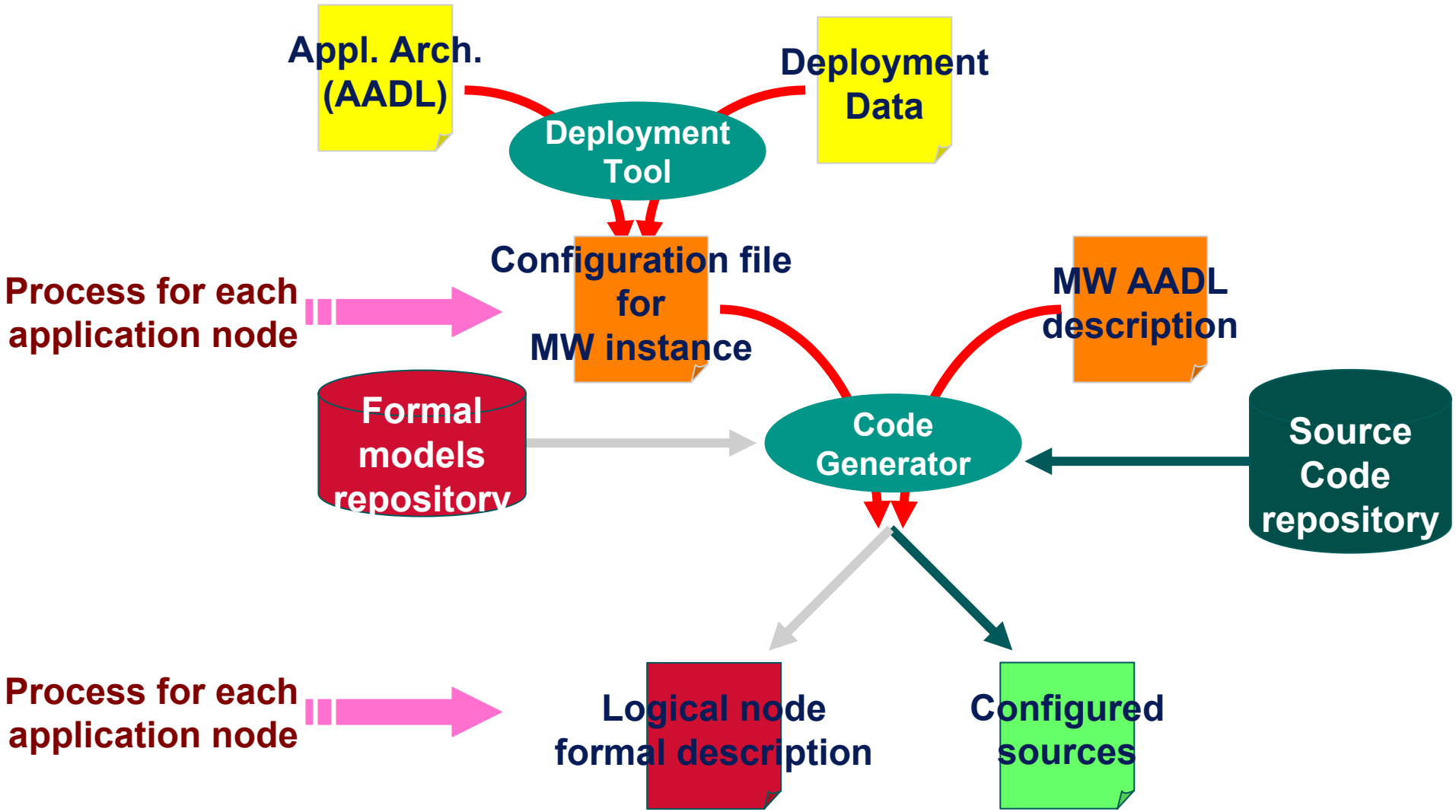
Model analysis: Generation of Petri Nets

- The AADL specifications can be used to produce formal modeling of systems analysis on execution flows
  - detect structural errors in component assembling
  - Using CPN-AMI P/N toolsuite
- Other aspects can be handled by other formalisms and tools
  - schedulability
  - memory footprint requirements

```
process implementation prog.i
subcomponents
  thread1 : thread th1;
  thread2 : thread th2;
connections
  event data port th1.s -> th2.e;
end prog.i;
```



Unified Tool Chain

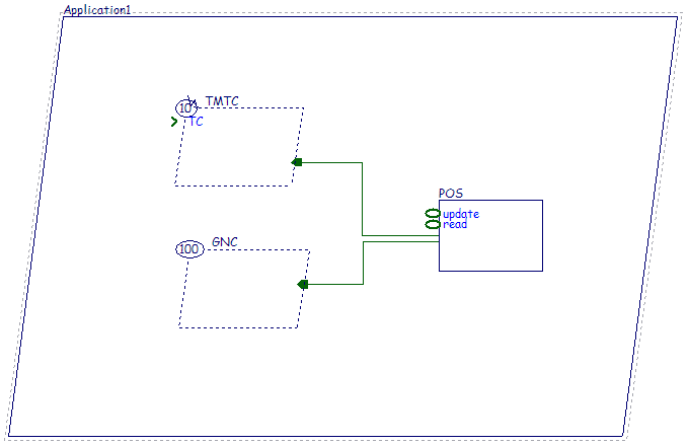


Toy Example for AADL

- Taken from IST-ASSERT project, from space domain
- 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 << 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

Playing with the Toy Example

- Cover typical interaction patterns
  - Periodic, aperiodic, shared variable
- Refinement from high-level view to deployment
- Serve as a basis to validate the process
- Down to code generation
  - System is schedulable
  - Has no deadlock
  - Can be generated for PolyORB-HI/QoS
  - Fully respects all compile-time restrictions
  - Then runs on tsim (LEON2 simulator)
- Ready for certification & deployment
  - System comes with both models and code
- Unified process for DRE systems
  - Either large set of QoS
  - Or HI restrictions



+

```
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;
```

Conclusion and Ongoing Work

- Building a DRE is still a complex task
  - Many configuration points, difficult to select them
  - Even more difficult to understand the interaction of 100+ policies
- Solution: integrating architecture analysis & DRE in one process
  - Modelling the system, verifying it, and then generating code
  - Respect separation of concerns between models, tools and middleware
- PolyORB as supporting runtime environment
  - QoS: RT-CORBA, DDS for "big" targets
  - HI: for "small" targets, with strong HI constraints
- AADL & Ocarina
  - Generation for both targets + verification (PN) + scheduling (Cheddar)
- Ongoing work in the context of IST-FP6 ASSERT
- Perspectives
  - More properties for code generation (deployment, configuration, …)
  - More on modelling (distribution paradigms, temporal analysis, dependability)