Generating high-integrity systems with AADL and Ocarina

Jérôme Hugues, ISAE/DMIA
jerome.hugues@isae.fr

Outline

- **AADL crash course**
- The Ocarina project
- AADL to Ada: experiments in IST-ASSERT
- AADL to C: experiments in ANR Flex-eWare
- Some other features

AADL components

- **AADL model** : hierarchy/tree of components

- **AADL component:**
  - **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, …)

Component type/implementation

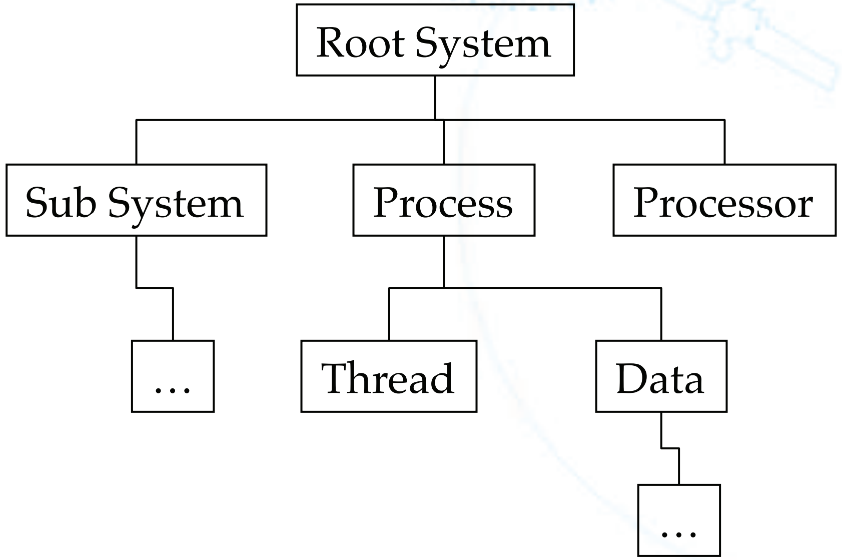
- AADLv2 distinguishes type and implementation

```
<category> foo
features
  -- list of features
  -- interface
properties
  -- list of properties
  --  e.g. priority
end foo;
```

```
<category> foo.i [extends <bar>]
subcomponents
  -- …
calls
  -- subprogram subcomponents
  -- called
connections
properties
  -- list of properties
  --  e.g. priority
end foo.i;
```

A full AADL system

- Component types and implementations only define a library of entities
- System must be instantiated through a hierarchy of subcomponents, from top-most (system) to top-down (subprograms, ..)
- Level N use entities at level N-1 as subcomponents, connect them



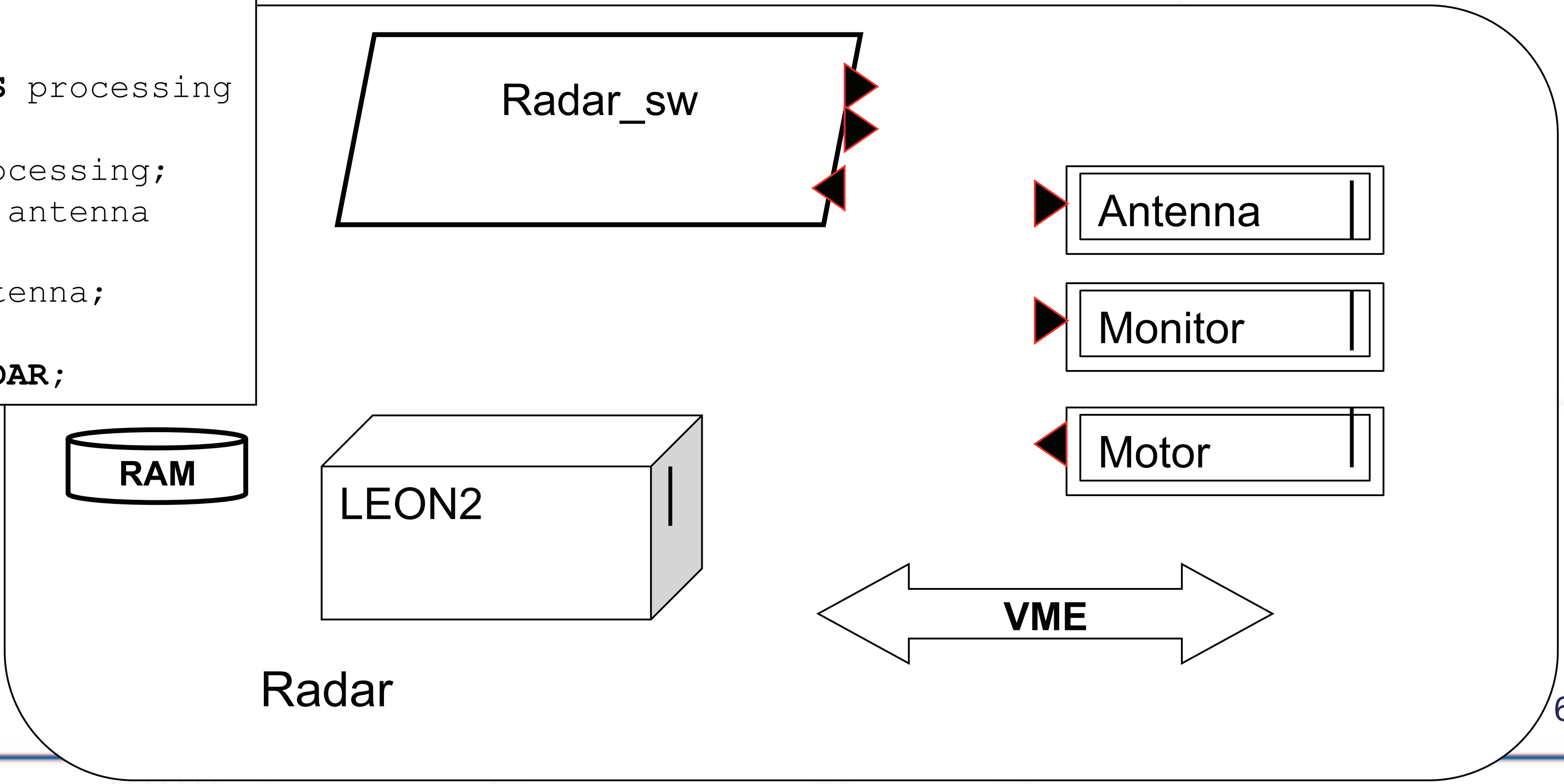
Radar case study

- Hardware/Software breakdown: components

```
PACKAGE radar
PUBLIC

PROCESS processing
-- …
END processing;
DEVICE antenna
-- …
END antenna;

END RADAR;
```

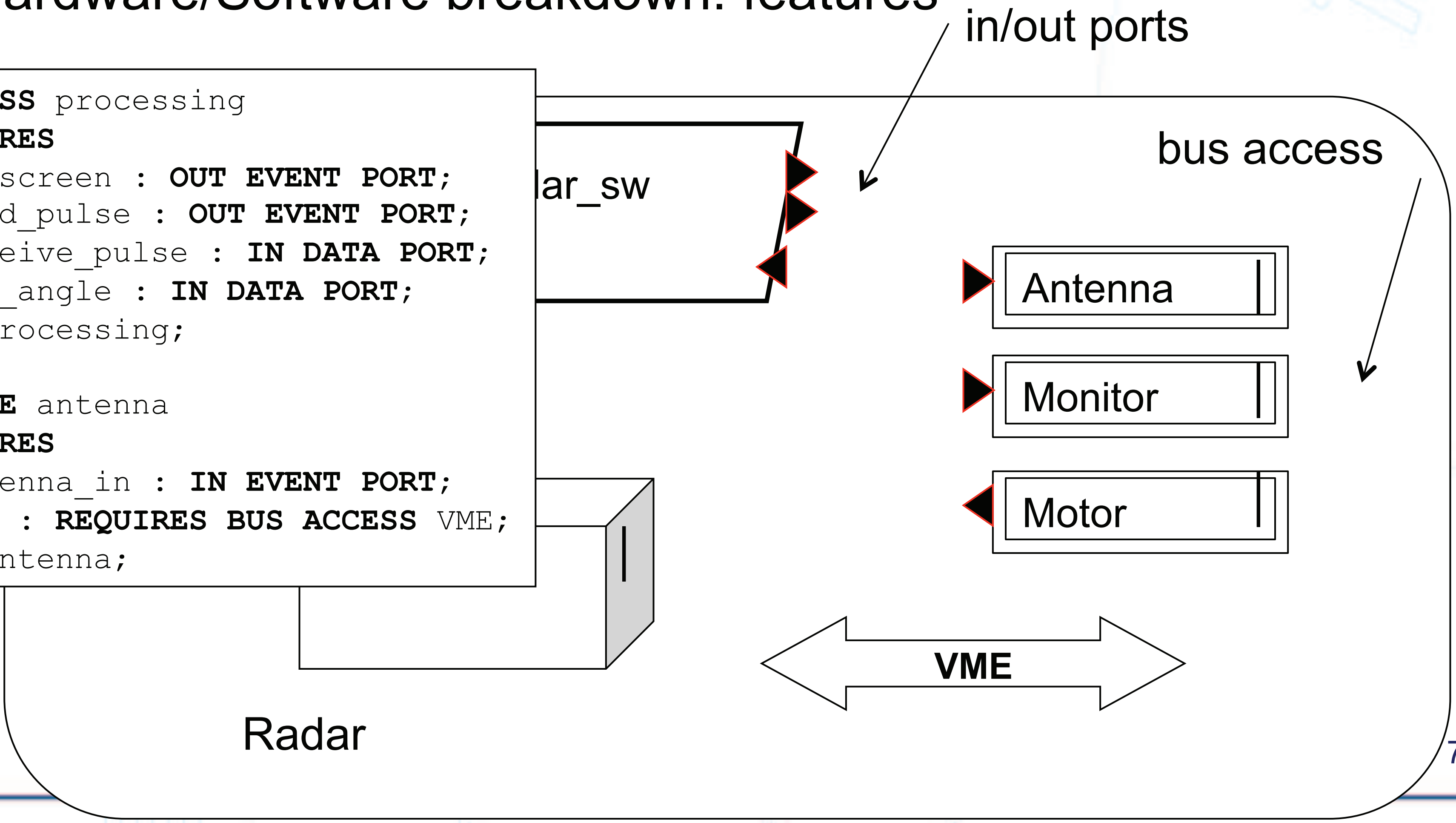


Radar case study

- Hardware/Software breakdown: features

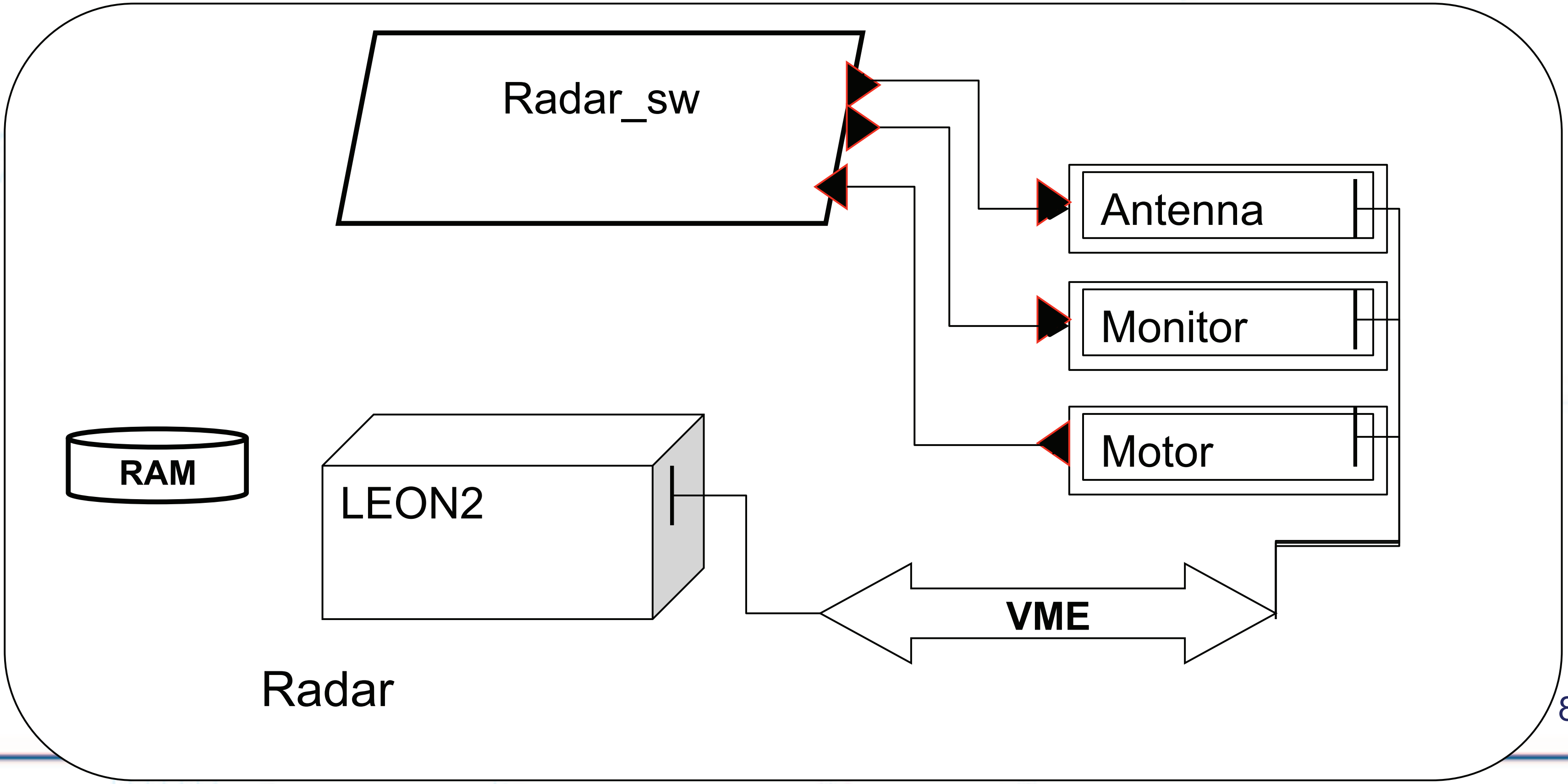
```
PROCESS processing
FEATURES
   to_screen : OUT EVENT PORT;
   send_pulse : OUT EVENT PORT;
   receive_pulse : IN DATA PORT;
   get_angle : IN DATA PORT;
END processing;

DEVICE antenna
FEATURES
   antenna_in : IN EVENT PORT;
   VME : REQUIRES BUS ACCESS VME;
END antenna;
```



Radar case study

- Hardware/Software breakdown: connections



Radar case study

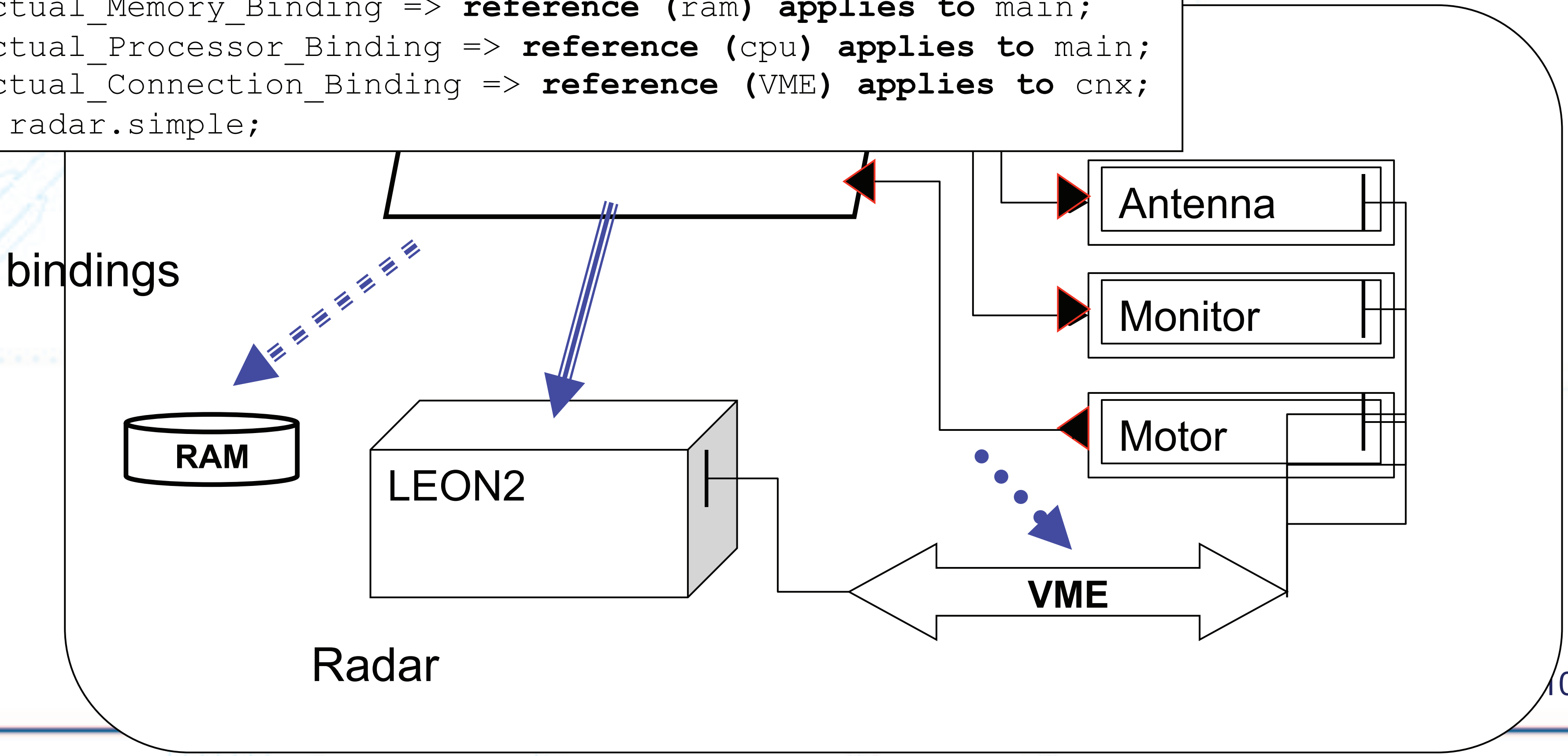
- Hardware/Software breakdown: connections

```
SYSTEM IMPLEMENTATION radar.simple
SUBCOMPONENTS
  aerial : DEVICE antenna;
  rotor : DEVICE motor;
  monitor : DEVICE screen;
  main : PROCESS processing.others;
  cpu : PROCESSOR leon2;
  VME : BUS VME;
  RAM : MEMORY RAM;
CONNECTIONS
  Cnx : PORT aerial.antenna_out -> main.receive_pulse;
  PORT rotor.motor_out -> main.get_angle;
  PORT main.send_pulse -> aerial.antenna_in;
  PORT main.to_screen -> monitor.screen_in;
  BUS ACCESS VME -> aerial.VME;
  BUS ACCESS VME -> rotor.VME;
  BUS ACCESS VME -> monitor.VME;
  BUS ACCESS VME -> cpu.VME;
  BUS ACCESS VME -> RAM.VME;
```

Radar case study

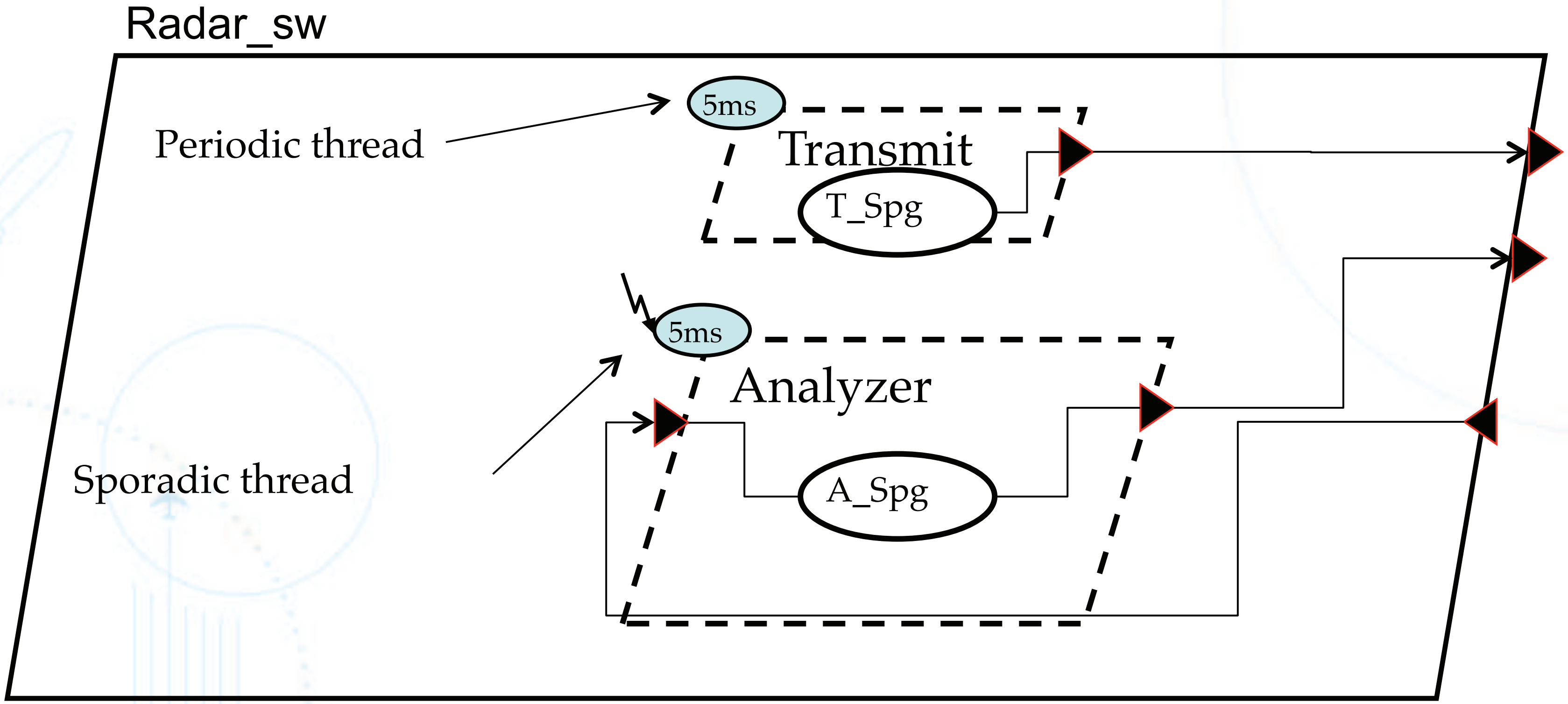
- Hardware/Software breakdown: bindings

```
PROPERTIES
  Actual_Memory_Binding => reference (ram) applies to main;
  Actual_Processor_Binding => reference (cpu) applies to main;
  Actual_Connection_Binding => reference (VME) applies to cnx;
END radar.simple;
```



Radar case study

- Software elements



Modeling with AADL, what else ?

- AADL is an interesting framework to model and validate complex systems: clear syntax, semantics, low overhead
  - "only" 300 pages for the core document
  - Increasing number of supporting tools for validation
  - MARTE standard to provide guidelines to model AADL patterns
- Scheduling analysis, resource dimensioning, behavior analysis, mapping for formal methods, fault analysis, …
  - Cheddar, Colored/Timed/Stochastic Petri Nets (CPN AMI, GreatSPN, TINA), FIACRE, BIP, Signal, Lustre, Alloy, TLA, UPPALL, Timed Automata, LOTOS
- AADL requirement document (ARD 5296)
  - Validate and **Generate** complex systems

Outline

- AADL crash course
- **The Ocarina project**
- AADL to Ada: experiments in IST-ASSERT
- AADL to C: experiments in ANR Flex-eWare
- Some other features

Ocarina: an AADL code generator

http://aadl.telecom-paristech.fr

- Ocarina is a stand-alone tool for processing AADL models
  - ➢ Command-line tool, a-la gcc
  - ➢ Can be integrated with third-party tools
    - ✓ OSATE (SEI), TASTE (ESA), Cheddar (UBO), MyCCM-HI (Thales)
    - ✓ Also emacs and vim modes
  - ➢ Joint work: Telecom ParisTech (leader), contributors ENIS, ISAE
- Fully supports both AADLv1 and AADLv2
- **Code generation** facilities target AADL runtimes
  - ➢ Ada HI integrity profiles, with Ada native and bare board runtimes
  - ➢ C POSIX or RTEMS, for RTOS & Embedded
  - ➢ C/ARINC653 and partitioned kernel POK
  - ➢ User code can be Ada, C, C++, Esterel, Simulink , Lustre, SCADE

Ocarina, other relevant features

- Model to model transformations
- WCET analysis of AADL runtime + user code: Bound-T
  - Take advantage on code generation patterns to “teach” how to measure WCET
- Constraint language to validate AADL model
  - Check static aspects of a system (see next presentation)
- Model checking models using Colored or Timed Petri Nets
  - Test for specific behavior scenarios
- Automatic evaluation of code coverage running scenarios
  - Based on the Couverture project
  - http://libre.adacore.com/libre/tools/coverage/

Ocarina distributions

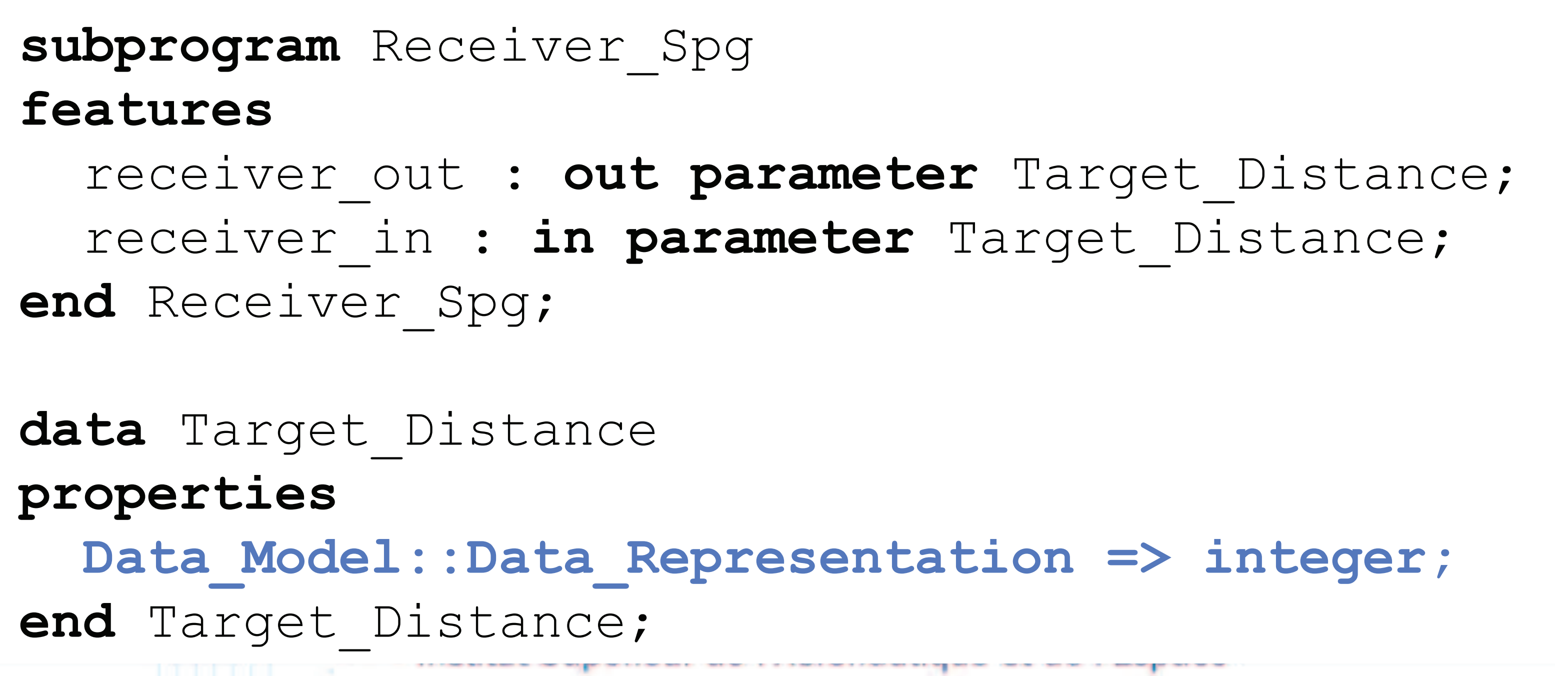
- http://aadl.telecom-paristech.fr/
- Ocarina 2.0 wavefront, daily snapshots
  - ➢ Binaries of Ocarina (release 1.2 and nightly builds)
    - ✓ For GNU/Linux, Windows, Solaris, Mac OS X, FreeBSD
  - ➢ Documentation and examples (30+ available)
  - ➢ Scientific papers on the use of AADL
  - ➢ Teaching materials for Master degree
- PolyORB-HI AADL runtimes
  - ➢ Two versions: Ada 2005 and C/RT-POSIX
- POK AADL runtime
  - ➢ For MILS and IMA-like systems, using time and space partitioning

AADL and code generation

- AADL has a full execution semantics
  - ➢ Allow for full analysis
    - ✓ Scheduling, security, error, behavior
- **Issue:** what about the implementation ?
  - ➢ How to go to code
  - ➢ Preserve both the semantics and non functional properties ?
- **Solution:** enrich AADL with annexes documents
  - ➢ To describe application data
  - ➢ To detail how to bind code to AADL models

AADL: modeling data types

- **Issue:** how to model data types: an integer, a struct?
- **Solution:** Data Modeling annex document
  - Property set and design patterns for modeling data type
  - Closer to source code



```
subprogram Receiver_Spg
features
  receiver_out : out parameter Target_Distance;
  receiver_in : in parameter Target_Distance;
end Receiver_Spg;

data Target_Distance
properties
  Data_Model::Data_Representation => integer;
end Target_Distance;
```

AADL and subprograms

- **Issue:** how to bind user code ?
- **Solution:** default AADLv2 properties / AADL runtime

```
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;
```

AADL runtime

- **Issue:** how to interact with message queues ?
- **Solution:** use the AADL runtime (A.9) that define 10 services to interact with queues, …

```
subprogram Send_Output
features
    OutputPorts: in parameter <implementation-dependent>;
    -- List of ports whose output is transferred
    SendException: out event data;
    -- exception if send fails to complete
end Send_Output;
```

- Unfortunately, it remains implementation-defined
    - ➢ Mostly to allow for different designs, and enhance performances

AADL and programming languages

- **Issue:** how to map source code ?
- **Solution:** guidelines provided in the programming language annex document
  - ➢ Define mapping rules between AADL and the target language

```
subprogram Receiver_Spg
features
  receiver_out : out parameter Target_Distance;
  receiver_in : in parameter Target_Distance;
end Receiver_Spg;
```

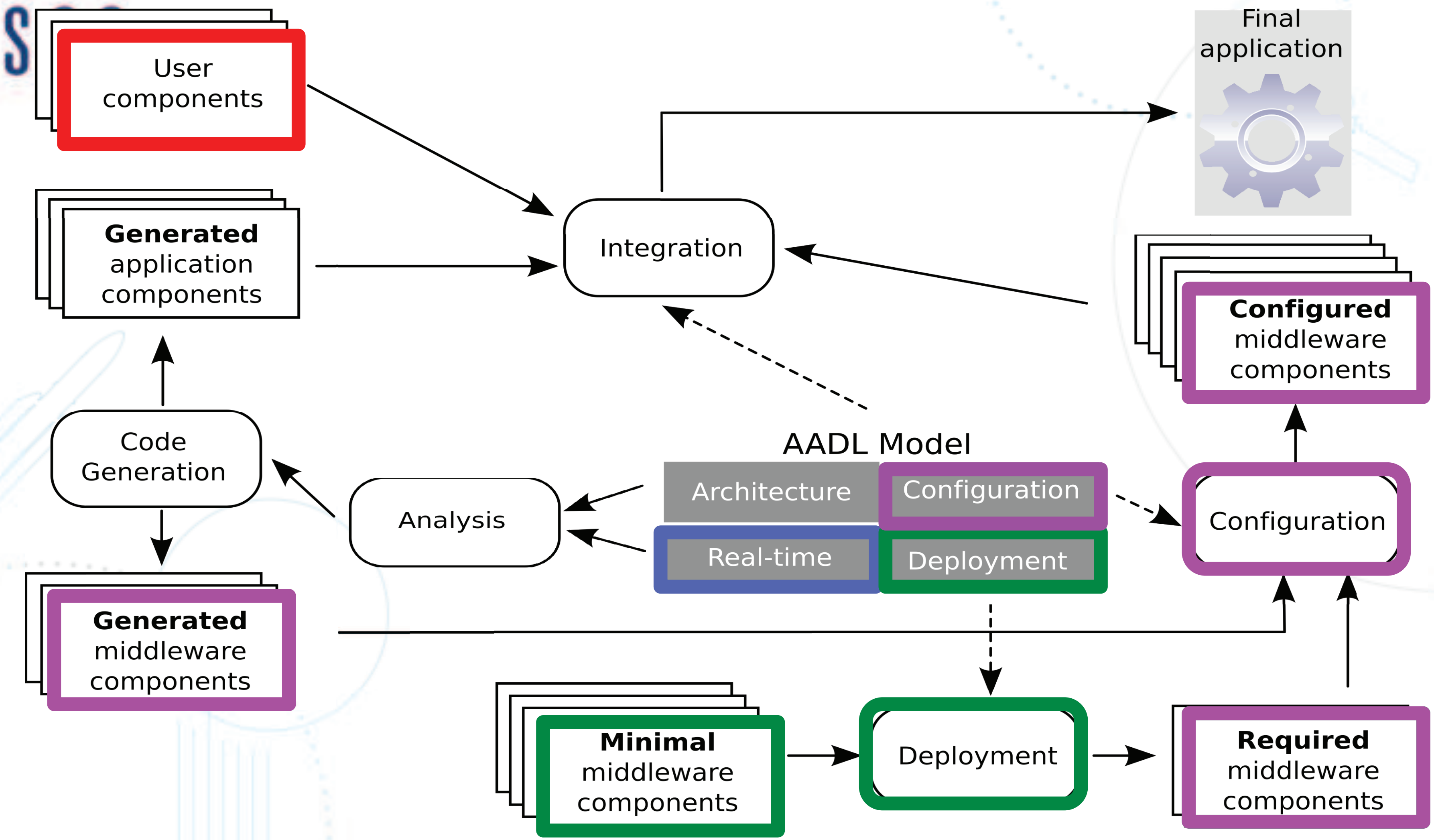
```
procedure Receiver
    (Receiver_Out : out Target_Distance;
     Receiver_In : Target_Distance);
```

```
void receiver
   (target_distance *receiver_out,
    target_distance Receiver_in);
```

AADL and code generation

- **Issue:** How much code should we write ? Tasks ? Queues ?
- **Answer:** the architecture says all
  - One can define a full framework and use it
    - ✓ Limited value, a-la CORBA
  - Generate as much as possible
- Ocarina: massive code generation
  - Take advantage of global knowledge to optimize code, and generate only what is required
  - Rely on a restricted runtime to support basic constructs

Building process for HI-DRE systems


User components
Generated application components
Integration
Final application
Configured middleware components
AADL Model
Architecture
Configuration
Real-time
Deployment
Code Generation
Analysis
Configuration
Generated middleware components
Minimal middleware components
Deployment
Required middleware components

Ocarina and code generated

- Strong emphasis on code quality
  - Generate code compatible with coding standards for HI systems
- Ada code: “easy”, checked by the compiler
  - Ravenscar profile for deterministic concurrency
  - HI restrictions: no dynamicity (OO, memory, …)
  - Also, simplifies the runtime, approx. 2200 SLOC
- C code: more tricky
  - Stringent coding guildelines for now
  - Consistent with ECSS-E-40A (ESA) and Thales practice
  - Even with POSIX: 2400 SLOC

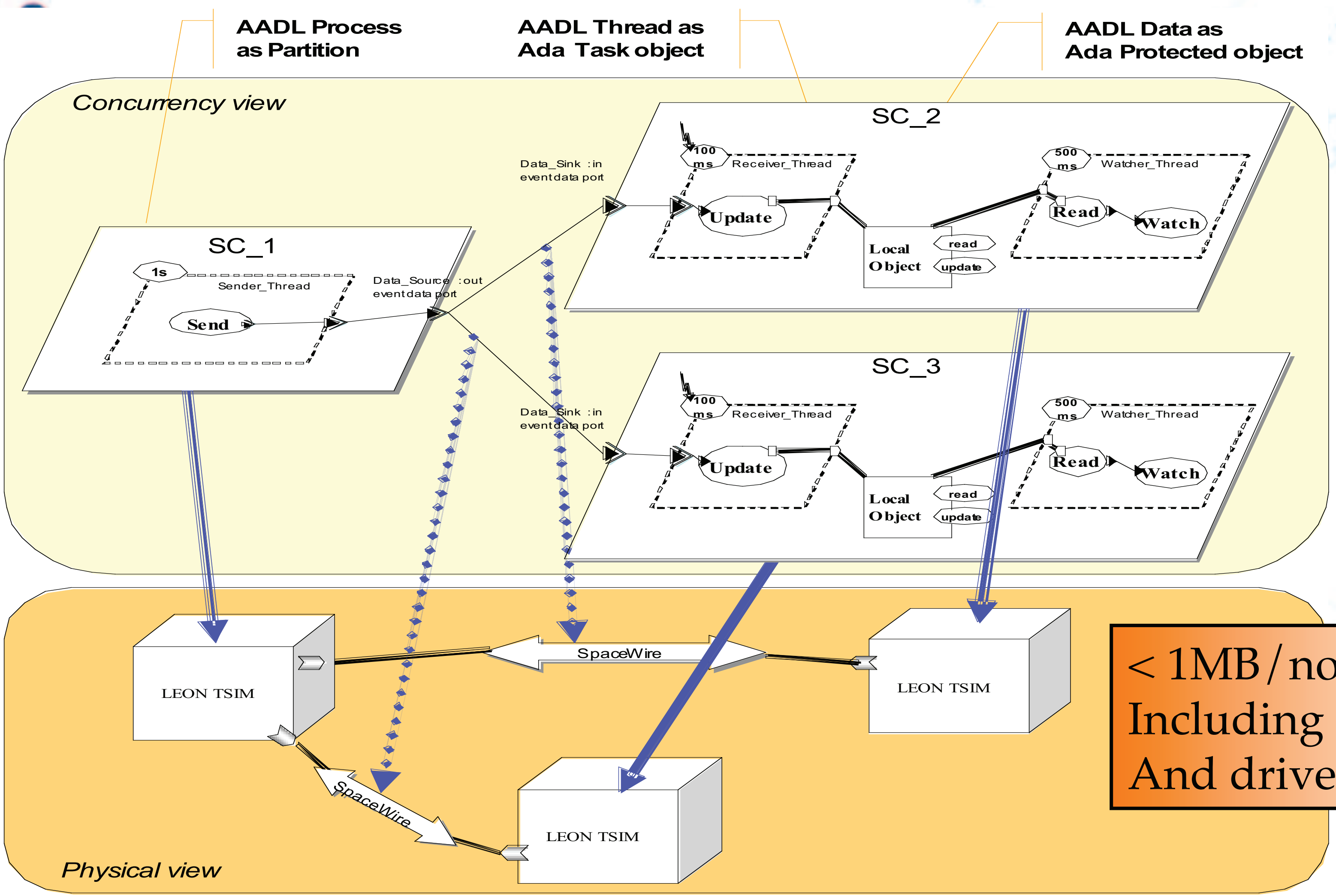
Outline

- AADL crash course
- The Ocarina project
- **AADL to Ada: experiments in IST-ASSERT**
- AADL to C: experiments in ANR Flex-eWare
- Some other features

Ocarina’s AADL runtime #1: Ada

- PolyORB-HI/Ada
  - Target Ada Ravenscar and High-Integrity runtimes
  - Supports AADL semantics, v1 and v2
  - Based on the Ravenscar & HI Ada profiles
    - ✓ Meets stringent requirements for High-Integrity systems, e.g. ESA
    - ✓ Checked at compile-time by Ada compiler, GNAT
    - ✓ On-going work to support SPARK/Ada
  - Supports native, RTEMS, and LEON2, ERC32 bare-board targets
- Validated in the context of the IST-ASSERT and TASTE projects with ESA
  - Increasing user base

The ASSERT MPC V2 demonstrator (2007)

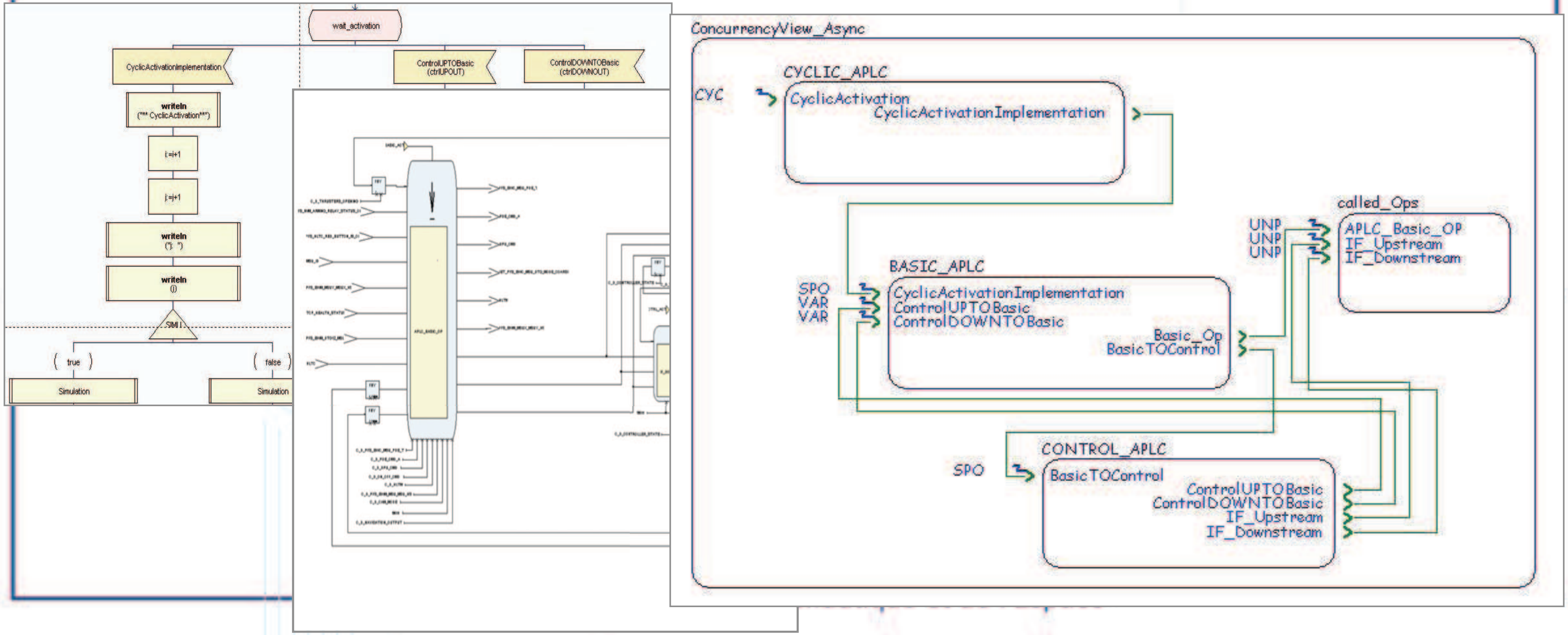


< 1MB/node,
Including RTOS
And drivers (60%)

The ASSERT ESA demonstrator (2008)

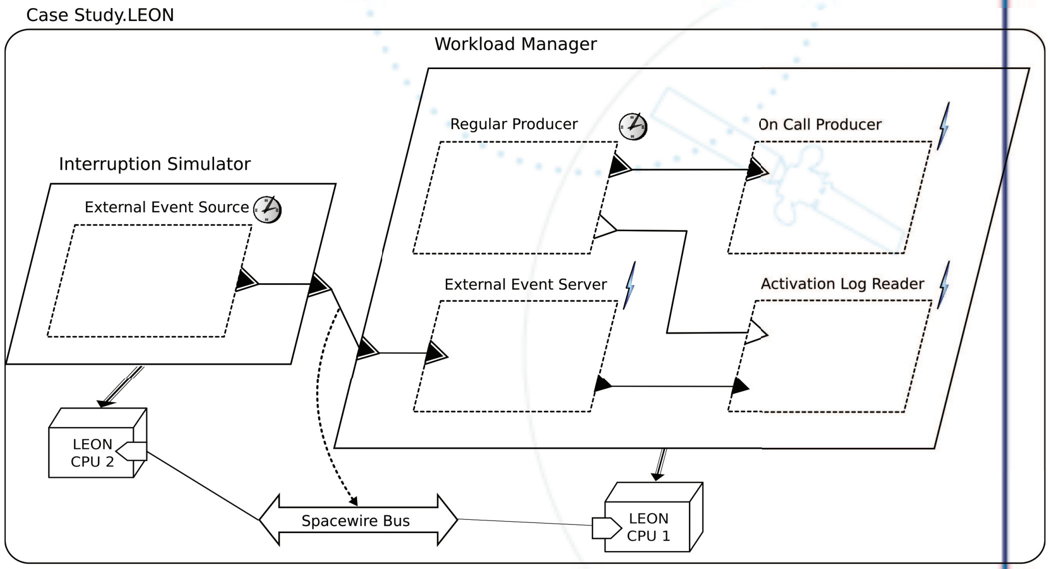
http://www.assert-project.net/

- Seamless integration of SDL, SCADE, Simulink, C, Ada, ASN.1, AADL
- Follow-up activities in TASTE: add VHDL, formal verifications



AADL vs. manual coding (2008)

- Example from the “Guide for the use of the Ada Ravenscar Profile in high integrity systems »
  - Typical example of RT system patterns
  - AADL generated code vs. Ada hand-coded
- Same functional model
  - Both are analyzable with RMA and RTA
  - Shares same code quality enforced by Ada compiler



- **For LEON2 targets**
  - Penalty of 6% in memory size, equivalent WCET
- **Big improvement in analysis**

Outline

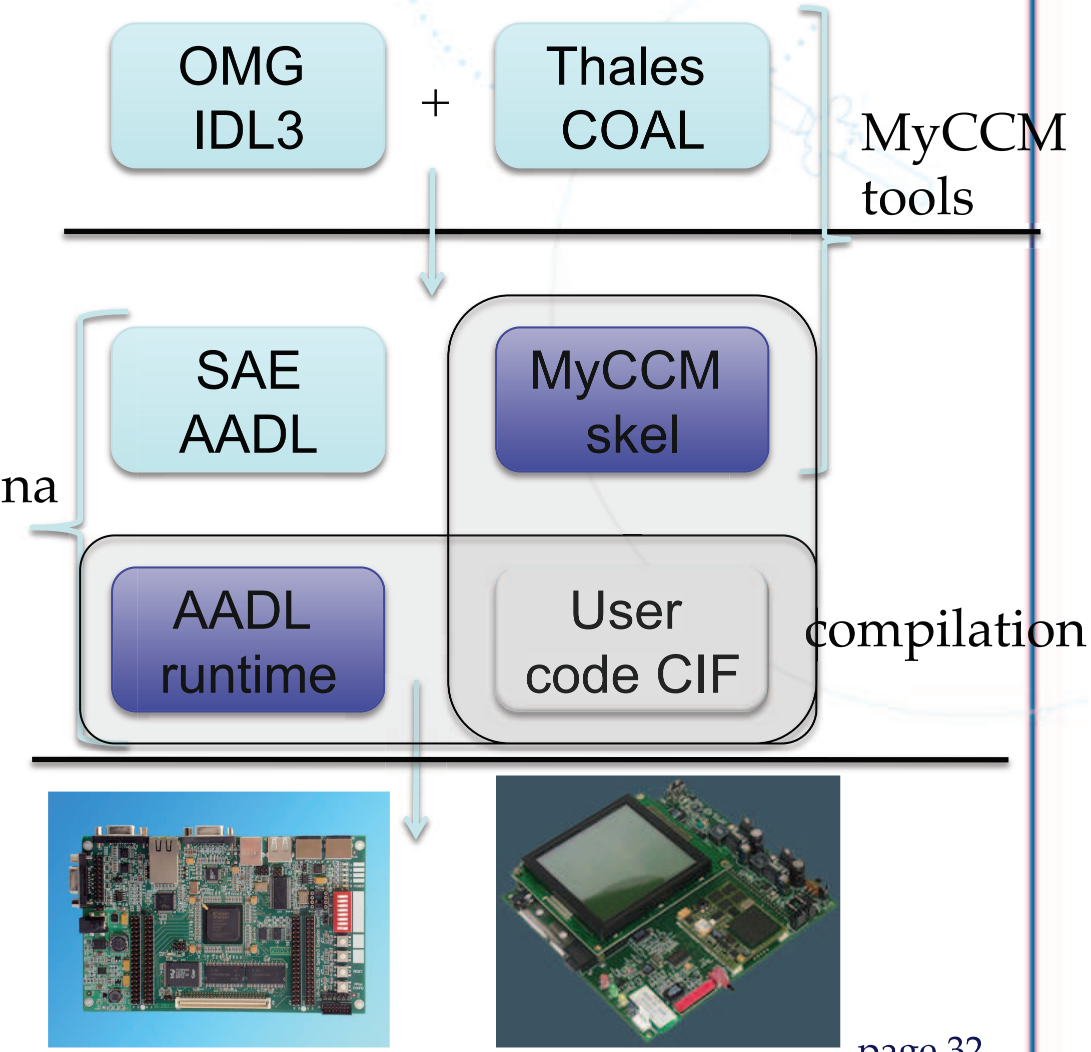
- AADL crash course
- The Ocarina project
- AADL to Ada: experiments in IST-ASSERT
- **AADL to C: experiments in ANR Flex-eWare**
- Some other features

Ocarina's AADL runtime #2: C/RT-POSIX

- PolyORB-HI/C
  - ➢ Targets C/RT-POSIX and C/RTEMS
    - ✓ Set of macros to support other RTOS
  - ➢ Tested on multiple operating systems
    - ✓ Native, GNU/Linux
    - ✓ Restricted libc: GNU/Linux on Nintendo DS and Nokia 770
    - ✓ POSIX RTOS: RTEMS
  - ➢ Tests demonstrated a limited subsystem of RT-POSIX & libc is enough to support AADL
  - ➢ Performance comparable to the Ada version
- Used in the ANR Flex-eWare project by Thales

Flex-eWare project (2009) Merging CCM and AADL

- Using ASSERT philosophy: combining notations
- LwCCM is interesting for system designers
  - ➢ Comfortable with the OMG
- Map onto AADL for consolidation
- Generate code using Ocarina
- Uses AADLv2


OMG IDL3
+
Thales COAL
MyCCM tools
Ocarina
SAE AADL
MyCCM skel
AADL runtime
User code CIF
compilation

Outline

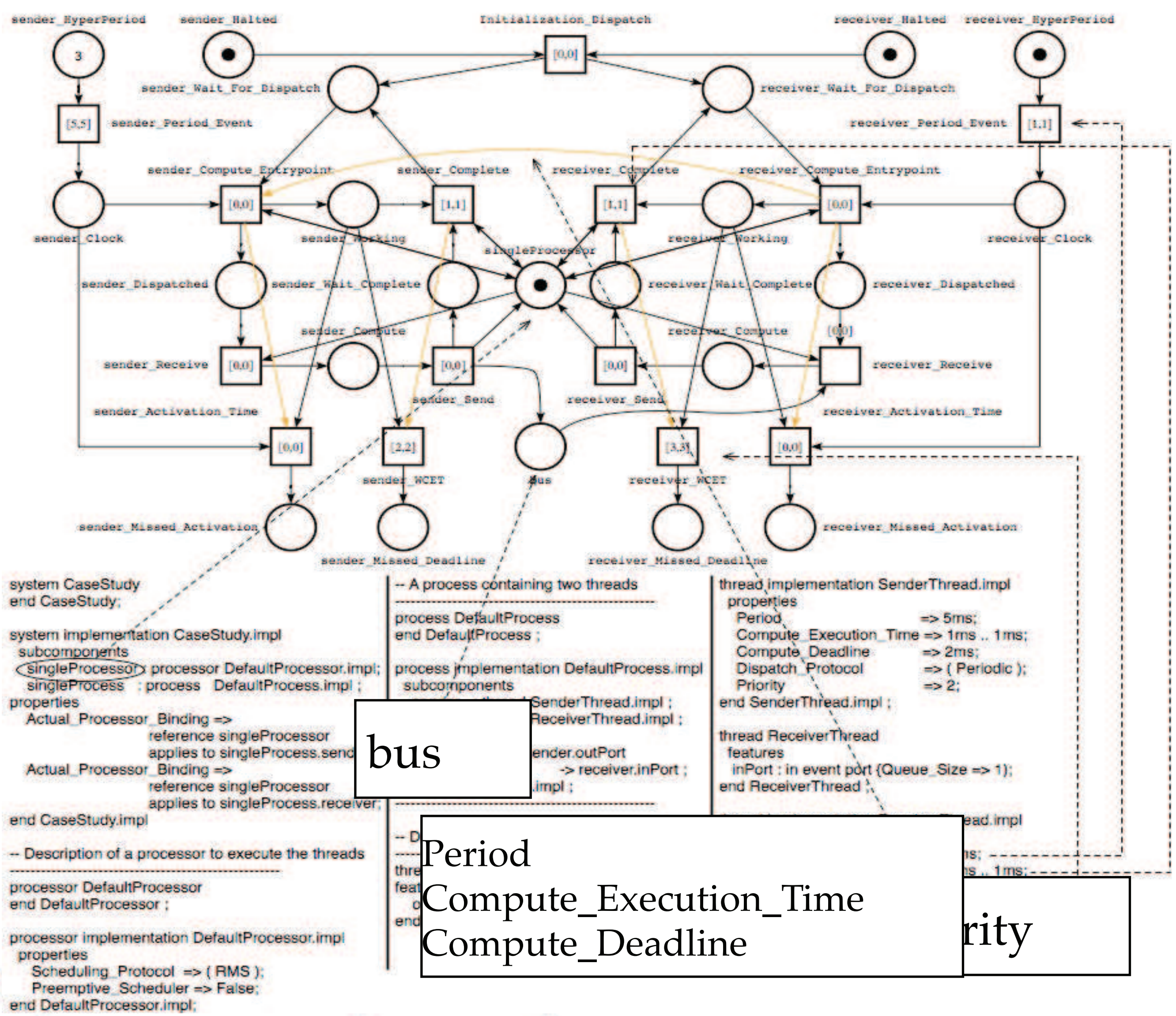
- AADL crash course
- The Ocarina project
- AADL to Ada: experiments in IST-ASSERT
- AADL to C: experiments in ANR Flex-eWare
- **Some other features**

Ocarina’s AADL runtime #3: IMA-like

- POK (http://pok.gunnm.org)
  - A bare board AADL runtime: both an AADL runtime and a kernel
  - Finely tuned using AADL properties
  - Follow ARINC philosophy for time and space partitioning
- Separate services as more as possible
  - Restrict functionalities of each service
  - Fine-grain configuration
  - Ex: include static scheduler, not RMS
- Configures resources of each layer
- Main goal : use ONLY needed functionalities
  - Help the certification process (cf. DO178B)
  - Low memory footprint

Petri nets and AADL

- Colored PN
  - CPN-AMI
- Time PN
  - TINA
- Adapt patterns to the property to be checked (observers, or reduced patterns)



```
system CaseStudy
end CaseStudy;

system implementation CaseStudy.impl
  subcomponents
   singleProcessor : processor DefaultProcessor.impl;
   singleProcess   : process   DefaultProcess.impl ;
properties
   Actual_Processor_Binding =>
                    reference singleProcessor
                    applies to singleProcess.send
   Actual_Processor_Binding =>
                    reference singleProcessor
                    applies to singleProcess.receiver;
end CaseStudy.impl

-- Description of a processor to execute the threads
-----------------------------------------------------
processor DefaultProcessor
end DefaultProcessor ;

processor implementation DefaultProcessor.impl
  properties
    Scheduling_Protocol  => ( RMS );
    Preemptive_Scheduler => False;
end DefaultProcessor.impl;
```

```
-- A process containing two threads
---------------------------------------------
process DefaultProcess
end DefaultProcess ;

process implementation DefaultProcess.impl
  subcomponents
                     SenderThread.impl ;
                     ReceiverThread.impl ;

                     ender.outPort
                        -> receiver.inPort ;
                     .impl ;
---------------------------------------------

-- D
----
thre
feat
  o
end
```

```
thread implementation SenderThread.impl
  properties
   Period                 => 5ms;
   Compute_Execution_Time => 1ms .. 1ms;
   Compute_Deadline       => 2ms;
   Dispatch_Protocol      => ( Periodic );
   Priority               => 2;
end SenderThread.impl ;

thread ReceiverThread
  features
   inPort : in event port {Queue_Size => 1};
end ReceiverThread ;

                                  ead.impl

                                  ns;
                                  ns .. 1ms;
```

bus

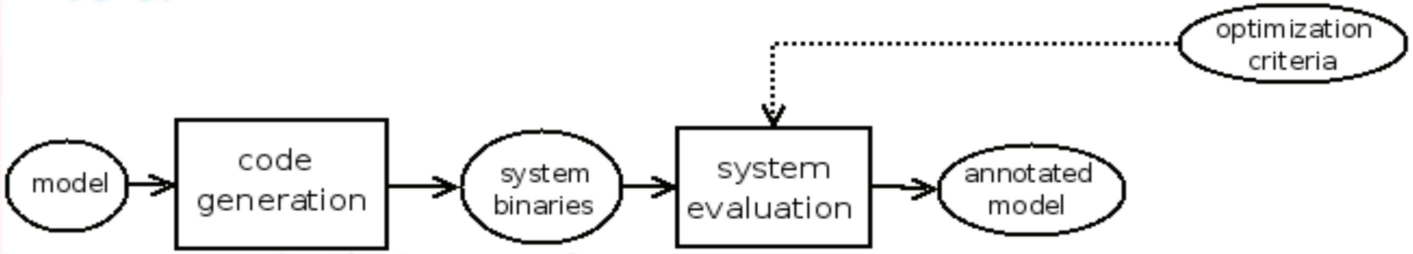
Period
Compute_Execution_Time
Compute_Deadline

rity

Institut

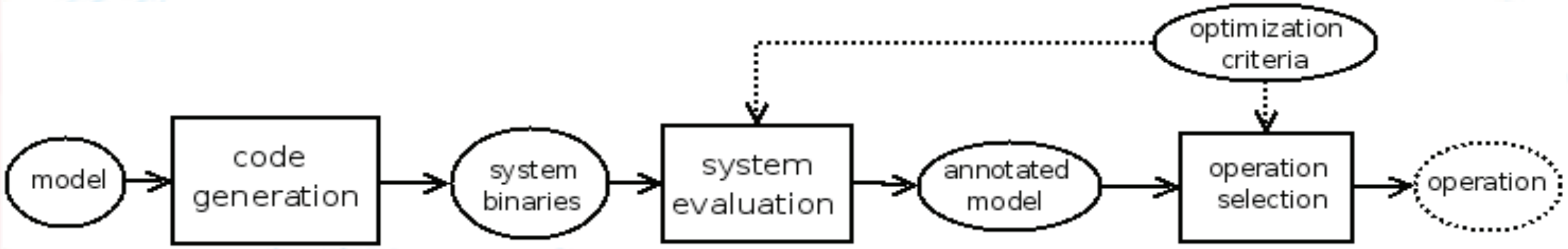
Optimizing AADL models

- Take advantage of full MBD chain to generate code and then evaluate system
  model-level evaluation: some user-defined metrics
  binary-level evaluation: WCET, binutils, …



Optimizing AADL models

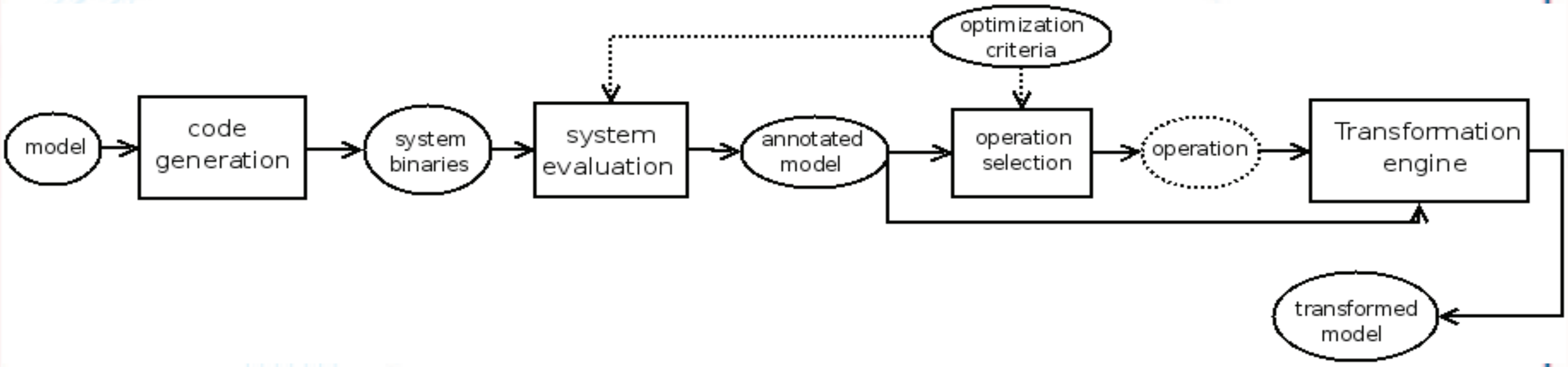
- Drive Optimisation process using REAL as a DSL to express relevant criteria
  As many criteria as projects



```
theorem minimum_distance_to_deadline
  foreach th in Thread_Set do
     var distance := if exists(th, "Transformations::Fusion_Occurred")
                     then compute distance_to_deadline_optimized (th)
                     else compute distance_to_deadline_regular (th);
  return (Mmin (distance));
end minimum_distance_to_deadline;
```

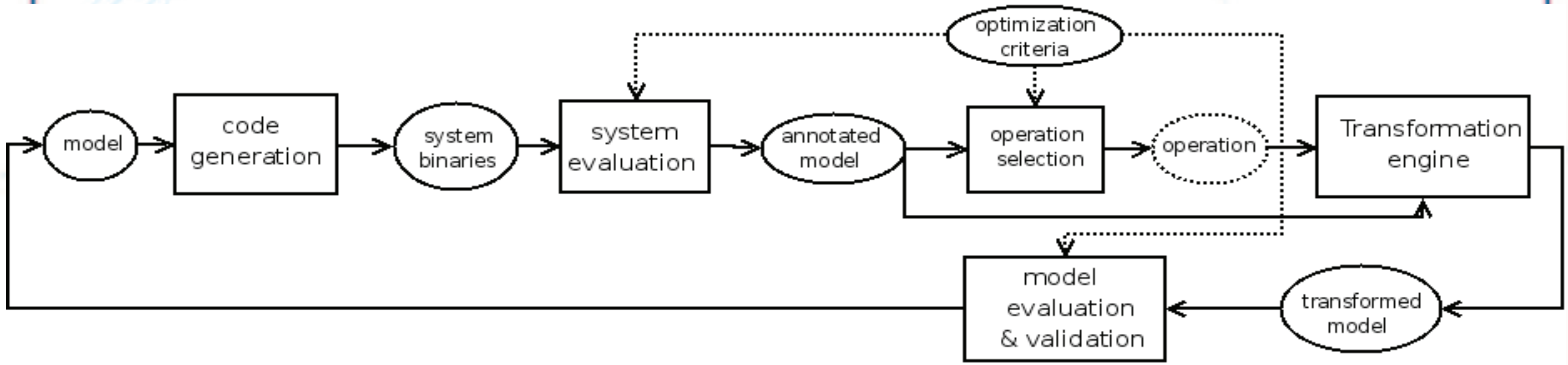
Optimizing AADL models

- Then perform the transformation

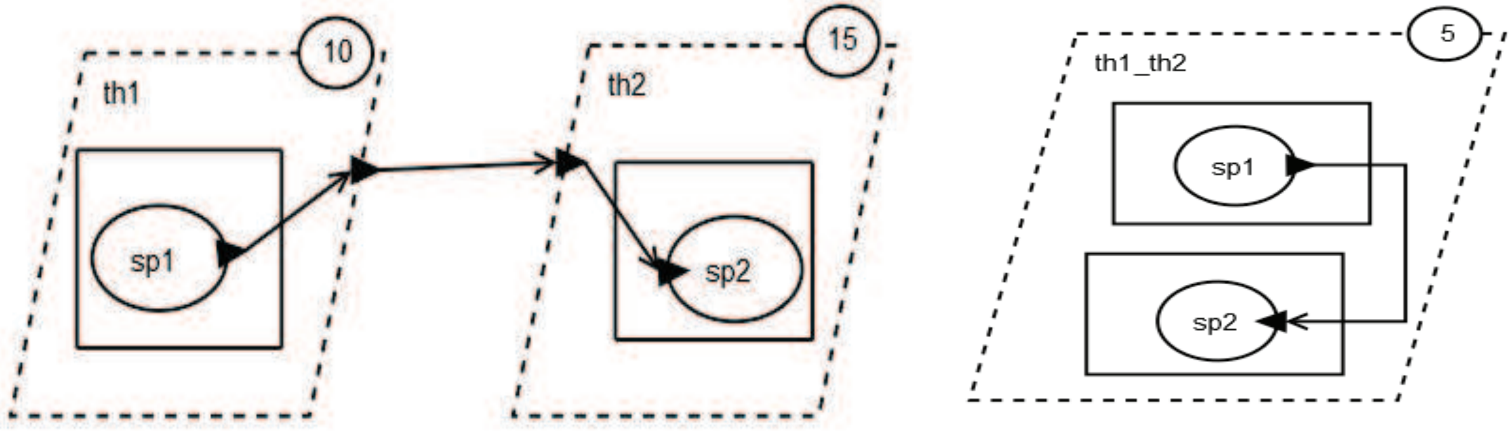


Optimizing AADL models

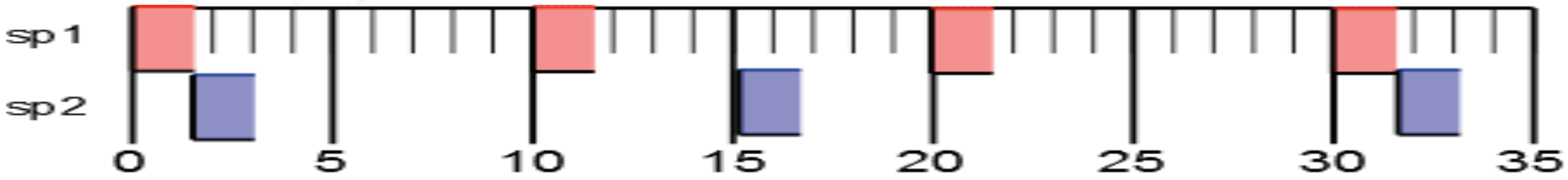
- And reiterate, up to your selected end point, or global minimum for your criteria



Optimizing AADL models: ex Merge



- Two periodic threads of periods 10 and 15 ms
  - Connected through a data connection (asynchronous)
- Merge : a periodic thread of period 5 ms
  - The tasks are connected through local connection



To conclude

- Ocarina provides tools to generate part of your system, and to relieve you from misconfiguration of the runtime
- Not presented
  - ➢ REAL: a constraint language to check properties on system
    - ✓ E.g. Bell-LaPadula, Biba, ARINC consistency, …
  - ➢ Bound-T integration: compute WCET of AADL runtime
  - ➢ Behavioral annex
  - ➢ Automatic execution of model: integrate compilation and run on simulator or real hardware in one click, to ease rapid prototyping
  - ➢ Code coverage of the model's generated code
  - ➢ …

Credits

- Ocarina is the result of more than 5 years of research
  - Lead work: Laurent Pautet (ENST) + Jérôme Hugues
  - Members of AS-2C since 2005
- PhD students involved
  - Thomas Vergnaud: initial architecture of Ocarina + code generation to PolyORB
  - Bechir Zalila: code generation to and design of PolyORB-HI/Ada
  - Julien Delange: PolyORB-HI/C + POK + ARINC 653
  - Xavier Renault: mapping to Petri Nets
  - Olivier Gilles : optimization of AADL models
  - Gilles Lasnier: integration of the Behavioral annex