Architectural Guidelines for the AADL to Enable the Automatic Generation of Systems

Thomas Vergnaud <thomas.vergnaud@enst.fr>

Irfan Hamid <irfan.hamid@enst.fr>

Scope of the AADL

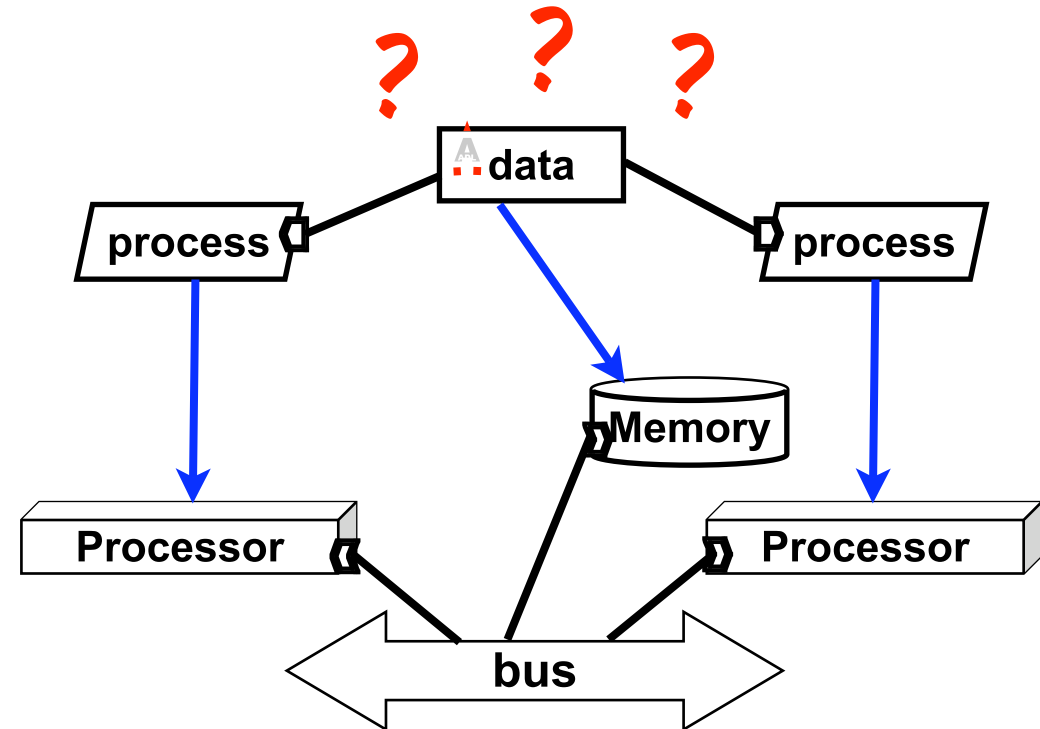
TELECOM PARIS
école nationale supérieure des télécommunications

- AADL components model parts of the actual systems
  - component with very clear semantics
    - *application components: thread, thread group, process, data, subprogram*
    - *platform components: processor, memory, device, bus*
  - "abstract" elements such as connectors are not very well handled by AADL

- AADL can be seen as a pre-implementation language
  - AADL descriptions reflect the actual system
  - MDA/PSM-level
    - *gather functional & non-functional characteristics*
      - components
      - properties
    - *depend on the implementation technology*

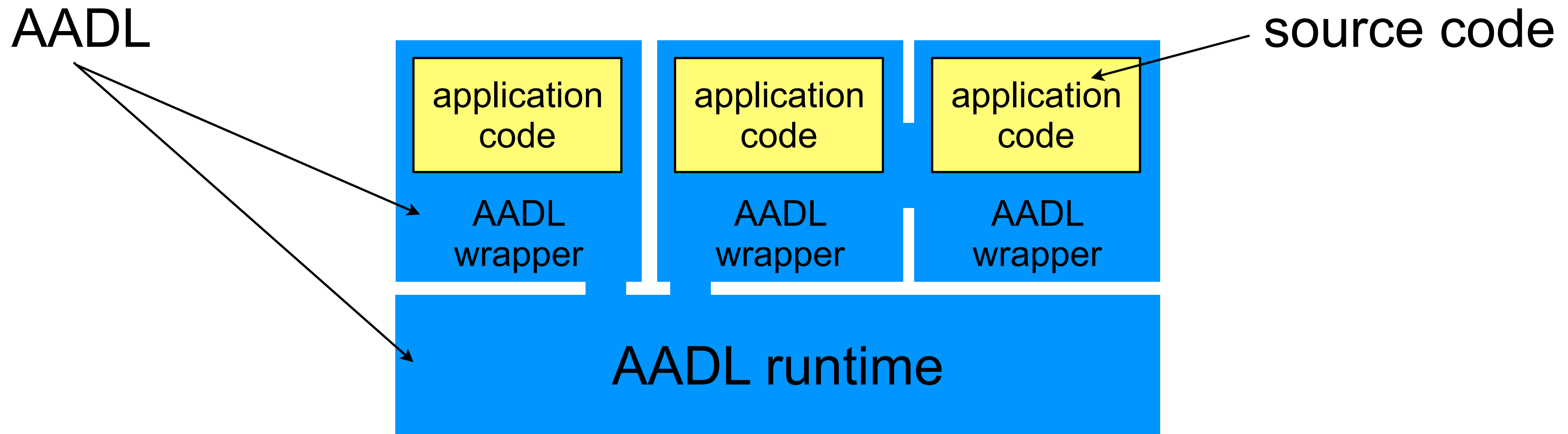
Assistance in the Design of AADL Architectures

- architectural constructions have to be turned into actual systems
- the AADL syntax may provide too much liberty in descriptions
  - need to specify which constructions can be implemented
  - need to define what is supported by the AADL runtime
- the AADL syntax allows specify many parameters
  - many properties
  - what kind of information are relevant for application generation?



Component-Based Applications

- Build a runtime that corresponds to what was specified in the AADL description
  - clear separation of architectural and algorithmic concerns
    - *source code: behavioral descriptions*
    - *AADL: runtime that controls the behavioral descriptions*
  - help analyze systems
  - facilitate reconfiguration



What Kind of Guidelines?

TELECOM PARIS
école nationale supérieure des télécommunications

- We focus on possibly distributed systems
  - distribution models
    - *communications between application nodes*
  - scheduling policies
    - *management of system threads*
  - node deployment
    - *locations of the nodes*
    - *runtime configuration from the environment description*
- Guidelines to structure architectural descriptions
  - runtime configuration: threads, processes, platform components
  - application: subprograms, data
  - allow the use of different runtime implementations

Architectural Patterns for Distribution Models

TELECOM PARIS
école nationale supérieure des télécommunications

- message passing
  - data ports, event data ports
  - messages are sent at the end of thread execution
- rpc
  - access to subprogram (AADL 1.2)
- distributed objects
  - access to data that provide subprograms

```
data d end d;

thread receiver_node
features
  e : in event data port d;
end receiver_node;

thread sender_node
features
  s : out event data port d;
end sender_node;
```

```
subprogram object
features
  method : subprogram sp;
end object;

thread server_node
features
  remote : provides data access object;
end server_node;
```

```
subprogram sp end sp;

thread server_node
features
  rpc : provides subprogram access sp;
end server_node;

thread client_node
features
  rpc : requires subprogram access sp;
end client_node;
```

Architectural Patterns to Structure Applications

TELECOM PARIS
école nationale supérieure des télécommunications

- behavioral descriptions
  - standard properties
- data type semantics
  - AADL properties in Language_Support

```
subprogram sp
features
  e : in parameter a_data.integer;
end sp;

subprogram sp1 end sp1;
subprogram sp2 end sp2;

subprogram implementation sp.i
calls
  seq1 {call1 : subprogram sp1;};
  seq2 {call2 : subprogram sp1;
        call3 : subprogram sp2;
        call4 : subprogram sp1;};
properties
  Source_Language => Ada95;
  Source_Name => “Sp_Code”;
  Source_Text => (“repository.adb”);
end sp.i;
```

```
data a_data end a_data;

data implementation a_data.integer
properties
  Language_Support::Data_Format => Integer;
end a_data.integer;

data implementation a_data.integer_array
properties
  Language_Support::Data_Type => classifier a_data.integer;
  Language_Support::Data_Structure => Array;
  Language_Support::Dimension => (6);
end a_data.integer_array;
```

Architectural Patterns to Deploy Applications

TELECOM PARIS
école nationale supérieure des télécommunications

- node locations, port numbers
  - AADL properties
- platform & processes descriptions
  - appropriate communication protocols to use
  - compiler to use
  - scheduling policy to implement
  - optimizations for local interactions
- clear separation
  - thread descriptions (application nodes)
  - process instanciations (application deployment)

```
process a_process
end a_process;

processor a_processor
end a_processor;

system global
end global;

system implementation global.i
subcomponents
  prog : process a_process
   {runtime::port_number => 5600;
    runtime::compiler => gcc;};
  proc : processor a_processor
   {runtime::location => "127.0.0.1";
    runtime::architecture => i386;};
properties
  actual_processor_binding =>
    reference proc applies to prog;
end global.i;
```

Interpretation of the AADL Guidelines in the Scope of a Middleware-based System

Code Generation: Application Code

TELECOM PARIS
école nationale supérieure des télécommunications

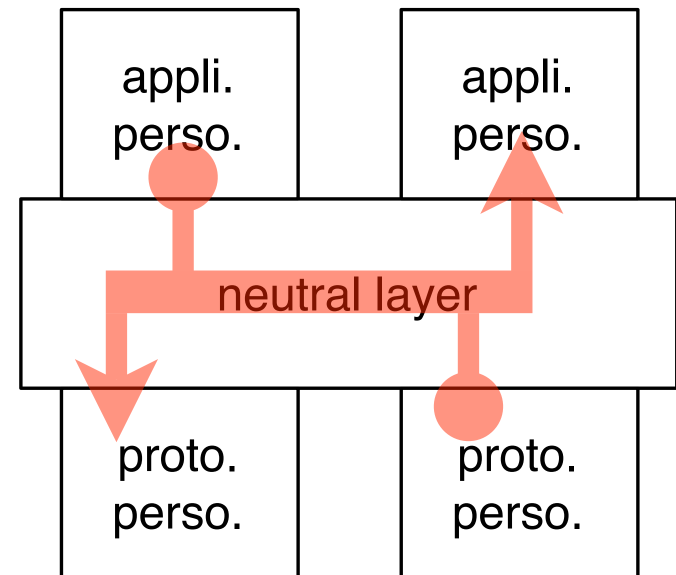
- to translate AADL components declarations
  - subprograms ➞ wrappers for source code
  - data components ➞ data type declarations
- various language mappings
  - Ada, C, Java…

```
data Integer_Type
properties
  Language_Support::Data_Format => Integer;
end Integer_Type;

subprogram Sp
features
  e : in parameter Integer_Type;
properties
  Source_Language => Ada95;
  Source_Name => "Sp_Code";
  Source_Text => ("repository.adb");
end Sp;
```

```
with Repository;

type Integer_Type is new Integer;

procedure Sp (e : in Integer_Type);

procedure Sp (e : in Integer_Type) is
begin
  Repository.Sp_Code (e);
end Sp;
```

Code Generation: Use of a Distribution Runtime

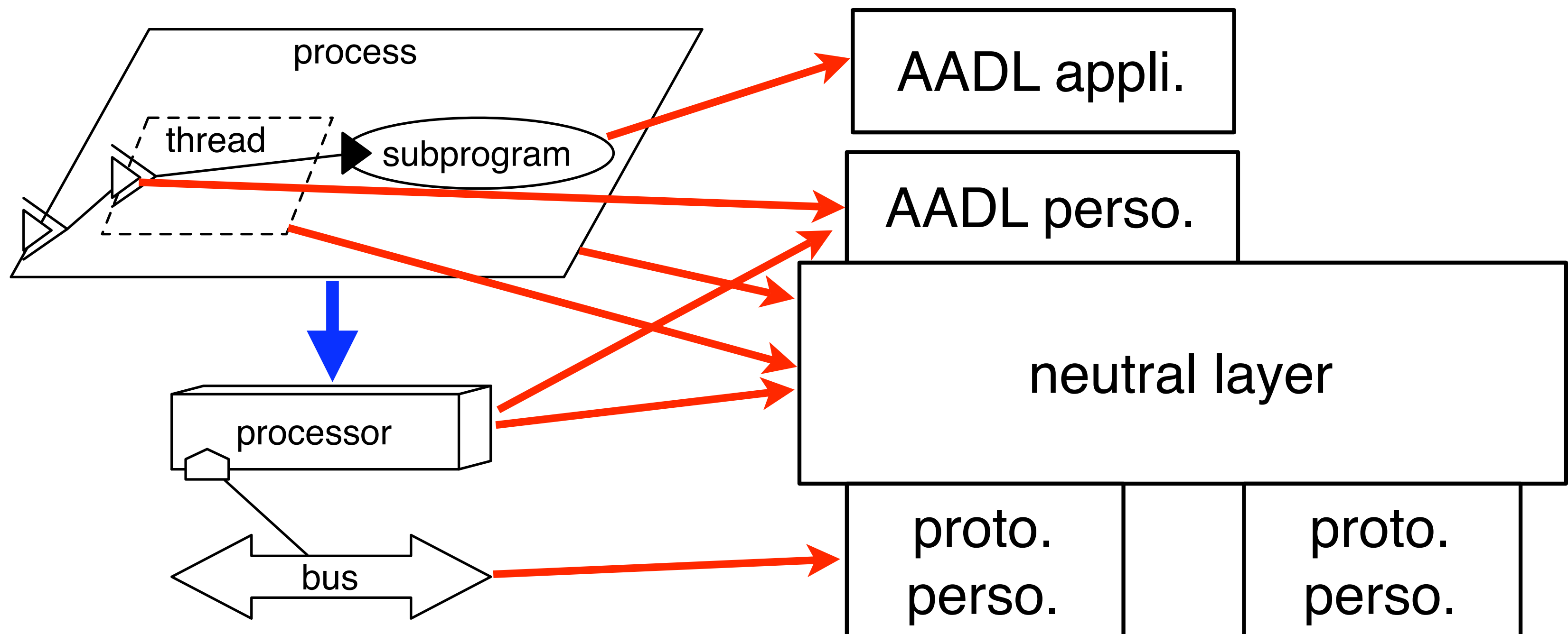
- Need for a highly configurable & reliable middleware
  - many distribution models to handle
  - many configuration parameters
  - must support formal verification
- PolyORB provides the necessary features
  - adaptable middleware framework
    - *different personalities to manage many protocols & APIs*
    - *neutral middleware core*
      - canonical architecture
      - selection & configuration of components
  - middleware core modeled using Petri nets
    - *ensure correct execution (no deadlock…)*



Code Generation: Runtime Configuration

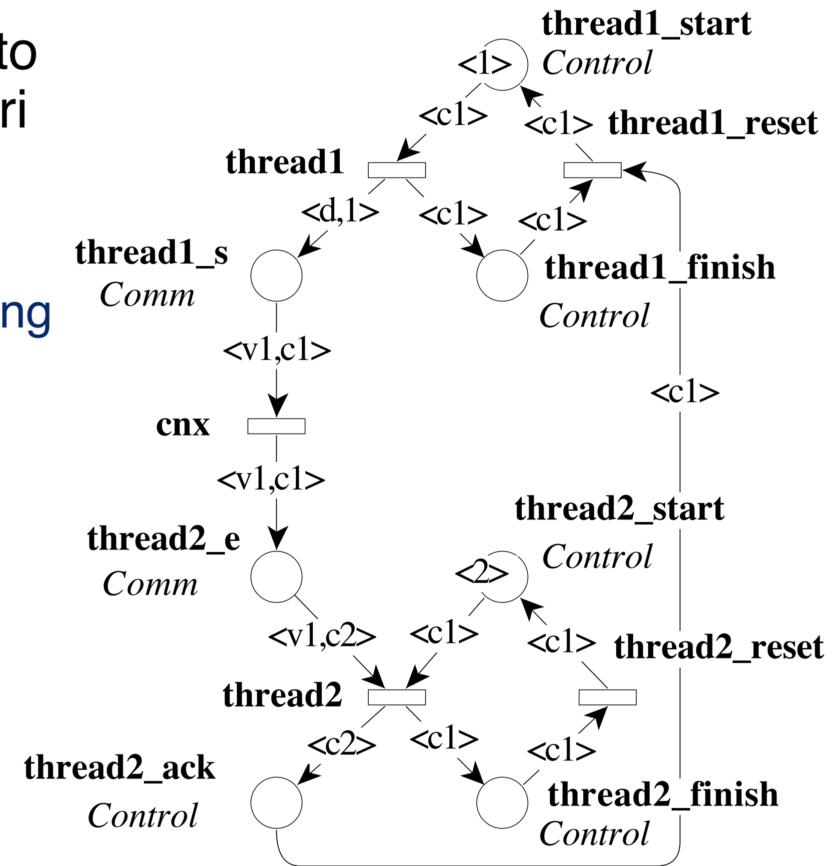
TELECOM PARIS
école nationale supérieure des télécommunications

- Architectural characteristics are translated into a configured runtime
  - AADL application personality
    - *from the thread features*
  - neutral core configuration
    - *from the organization of the application node*
  - selection of protocol personalities
    - *from the node environment*



Code Generation: Generation of Petri Nets

- The AADL runtime specifications can be used to produce formal modeling of systems using Petri nets
  - analysis on execution flows
  - detect structural errors in component assembling
- Other aspects can be handled with other formalisms
  - schedulability
  - memory footprint requirements
  - etc.




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

```
Class
  Control is 1 .. 2;
  Value is [d,u];
Domain
  Comm is <Value,Control>;
Var
  v1 in Value;
  c1, c2 in Control;
```

Conclusion on Distributed Systems Generation

TELECOM PARIS
école nationale supérieure des télécommunications

- Interpretation of the AADL architectural guidelines to use a middleware
  - several possible runtime implementations
  - source code generation is not affected by the runtime
- PolyORB as an AADL runtime
  - highly configurable
  - can be interoperate with other systems
  - reliable middleware
- Ocarina: generation tool for PolyORB-based AADL systems
  - free software
  - http://ocarina.enst.fr

TELECOM PARIS
école nationale supérieure des télécommunications

Interpretation of the AADL Guidelines in the Scope of an Ada/Reavenscar System

High-Integrity Ada

TELECOM PARIS
école nationale supérieure des télécommunications

- Ravenscar profile (adopted 2003)
  - Restricts Ada for safety reasons
  - Provides schedulability guarantees
  - Provides safety guarantees
    - *No deadlock*
    - *No livelock*
    - *No priority inversion*
  - Assures predictable behavior
- Concerns only the tasking features, for sequential features there is Ada SPARK

Restrictions

TELECOM PARIS
école nationale supérieure des télécommunications

- All tasks periodic or sporadic
- Dispatching policy FIFO_Within_Priorities
- Task set static (no task creation or termination during system execution)
- Inter task communication only via protected objects
- No rendezvous among tasks
- No dynamically created protected objects
- No use of Ada asynchronous task control primitives
    - Allows accurate estimation of WCET
- No relative delays
- Only 1 entry per protected object
    - Entry is like a conditional variable
- Only one task may wait at an entry
    - Eases schedulability analysis
- Priority ceiling locking protocol

Tasks

TELECOM PARIS école nationale supérieure des télécommunications

- Ada95 has its own runtime and allows declaration of tasks
- Implement AADL threads by instantiating Ada tasks
- To reduce generated source code we define a task type Ravenscar_Task

```
loop
   Next_Dispatch := Ada_Real_Time.Clock + Period;
   Dispatch;
   delay until Next_Dispach;
end loop;
```

Protected and Suspension Objects

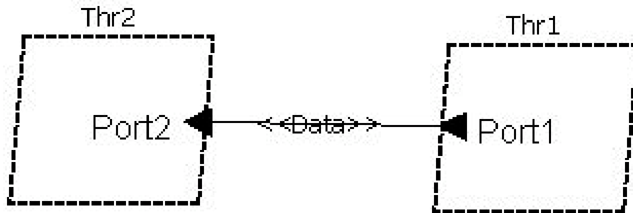
- A suspension object in Ada is like a simple conditional variable
  - Task A waits on suspension object s
  - Task B sets s to true, thus releasing A for one job
- A protected object is similar except that it may contain private data
  - Procedures to change internal state
  - Functions to read internal state
  - Entries to wait until a certain condition (barrier) becomes true
  - Protected objects may have at most one entry (Ravenscar)
  - Only one task can wait at entry of protected object (Ravenscar)

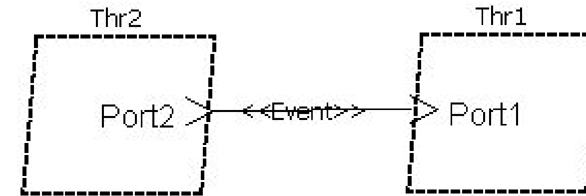
AADL Ports

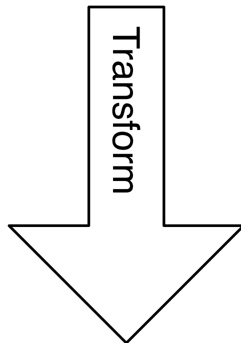
TELECOM PARIS
école nationale supérieure des télécommunications

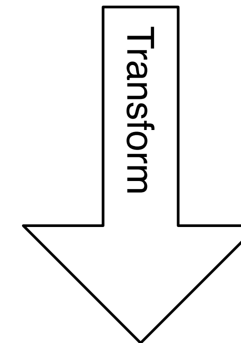
- AADL data ports are like shared memory
  - Transform AADL data ports into Ada protected objects without entries
  - We define a generic package that implements a simple protected object with a generic data type and get/set procedures
- AADL event ports are like pure signals in UML
  - They are queued upon reception
  - They may optionally launch special treatment
  - Transformed to either an Ada suspension object (with a counter, think counting semaphore) or a protected object with an entry (if multiple incoming event ports)
- AADL event data ports are like valued signals in UML
  - Their data is queued upon arrival
  - They may optionally launch special treatment
  - They are always transformed to an Ada protected object with an entry
  - We implement a generic package with a protected object that has an entry and standard get/set procedures
- Protected object/suspension object Selection rules
  - If a sporadic task has only one incoming event port then suspension object
  - In all other cases it is implemented as a protected object with an entry

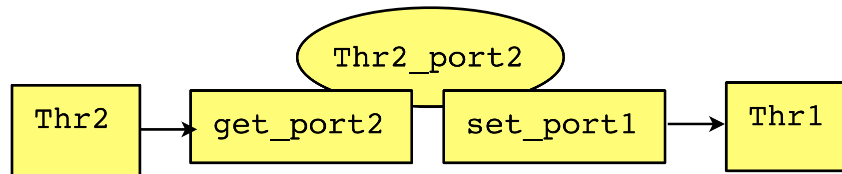
Task Communication

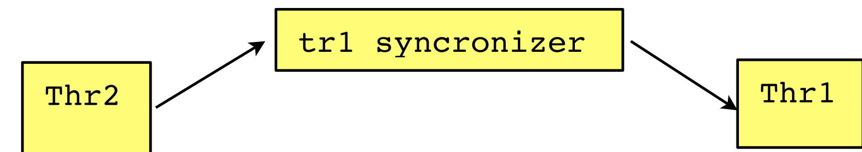




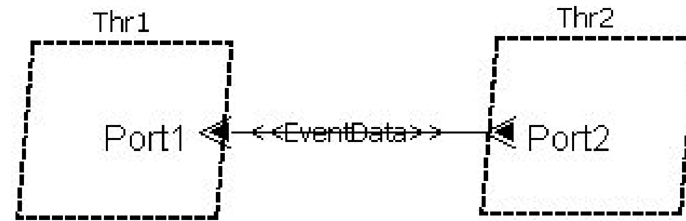


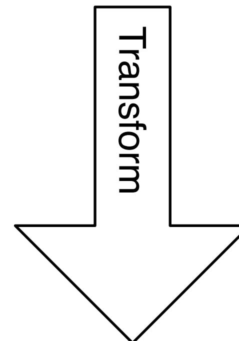


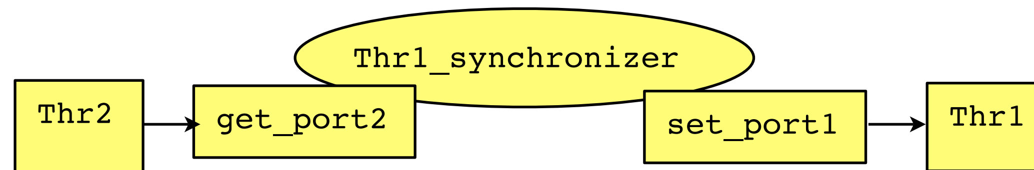




Task Communication (contd.)







Transforming a Periodic Task

TELECOM PARIS
école nationale supérieure des télécommunications

```
thread Thr1 end Thr1;
process Proc end Proc;

thread implementation Thr1.i
properties
  Dispatch_Protocol => Periodic;
  Period => 20 ms;
  Compute_Entrypoint => “My_Proc”;
end Thr1.i;

process implementation Proc.i
subcomponents
  Incoming : thread Thr1.i;
end Proc.i;
```

```
Incoming : Ravenscar_Cyclic_Task
  (xx, 20, My_Proc’access)
```

- Similarly the entire task set needs to be instantiated as Ada tasks
- For periodic tasks the thread’s own Compute_Entrypoint property is the procedure that is called for each job

Conclusion on AADL Ravenscar Generation

TELECOM PARIS
école nationale
supérieure des
télécommunications

- Ravenscar does not allow RPC
- All communications are asynchronous
- Distribution
  - Ravenscar doesn’t rule out distribution
  - ENST working on a lightweight real-time distribution middleware for Ravenscar
- Tooling being conducted at ENST
  - Take an AADL description and transform it into Ada Ravenscar code
  - MDD approach, transformation from AADL to Ada
  - Use of the standard AADL meta-model as starting point
  - Domain specific (Ravenscar) meta-model as intermediate step
  - Allow us to generate code for other Ravenscar runtimes (Java, C?)
- Investigation being carried out in the context of the ASSERT project

General Conclusion

TELECOM PARIS
école nationale supérieure des télécommunications

- Set of AADL guidelines
  - Help design implementable systems
  - Help define semantics for AADL runtimes
- Can be interpreted in different ways
  - Middleware-based systems
  - Specific Ada/Ravenscar runtime
- Associated with runtime semantics
  - AADL descriptions can be used to extract various formal representations