Generating Petri Nets from AADL descriptions

Thomas Vergnaud <thomas.vergnaud@enst.fr>

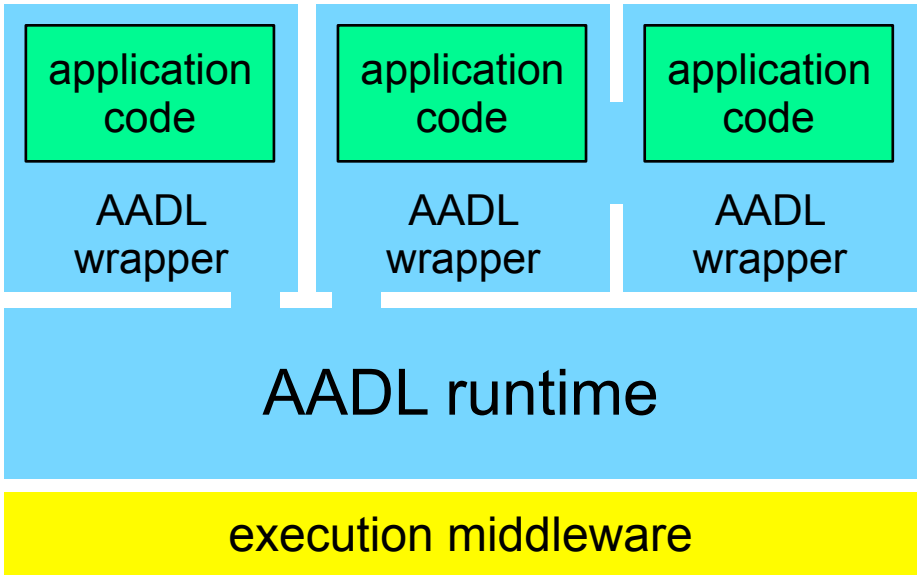
Designing a Distribution Runtime for the AADL

TELECOM PARIS
école nationale supérieure des télécommunications

- work at ENST focuses on the building of communication middlewares
  - manage communication between several application nodes
  - provide facilities to configure the middleware
- PolyORB: a framework to build adapted middlewares
  - collaboration with University Pierre & Marie Curie, in Paris
- How to ease the middleware configuration?
  - use a description of the application to capture the configuration parameters
  - AADL is a good solution for that

Global Architecture

- a runtime structure is created from the AADL description
  - built according to the AADL properties & architectural elements
- everything is on top of the middleware
  - handle the communications
  - manage the AADL threads



Integration of Source Code

TELECOM PARIS
école nationale supérieure des télécommunications

- Source code = behavioral description
- encapsulated inside AADL components
  - do not have visibility outside their component
  - do not control the runtime
- the AADL description provide all the information to run execute the behavioral descriptions
  - interfaces
  - location (binding to processors)
  - dispatch protocols and periods
  - etc.
- all the execution flows are described using the AADL
  - mappings between AADL and source code

Specifications of the Execution Middleware

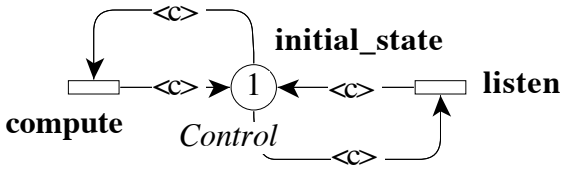
TELECOM PARIS
école nationale supérieure des télécommunications

- guidelines to help the user in describing an application that can be generated
  - distribution models (message passing, rpc, distributed objects)
  - clear separation application/runtime = subprograms/threads
- ensure we can produce compilable code from the architecture
- but does NOT provide any garantee concerning the executability
  - deadlocks?
  - undefined values?
- idea: use formal methods to perform verify the architecture before the actual code generation

Petri Nets

- formalism for model checking
  - “simulation” of all the states of an application
  - search for unwanted states
  - test the sequence of some states

- Petri nets are place/transition oriented graphs
  - tokens can circulate through the graph
  - can be stored in places
  - consumed/produced by transitions
  - initial marking = where the token are in the first state of the network

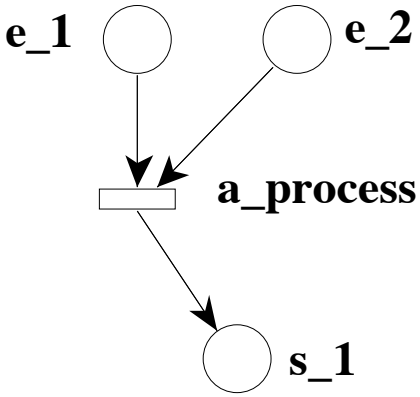



```
Class
 Control is 1 .. 1;
Var
 c in Control;
```

Basics of the Petri Net Transformation

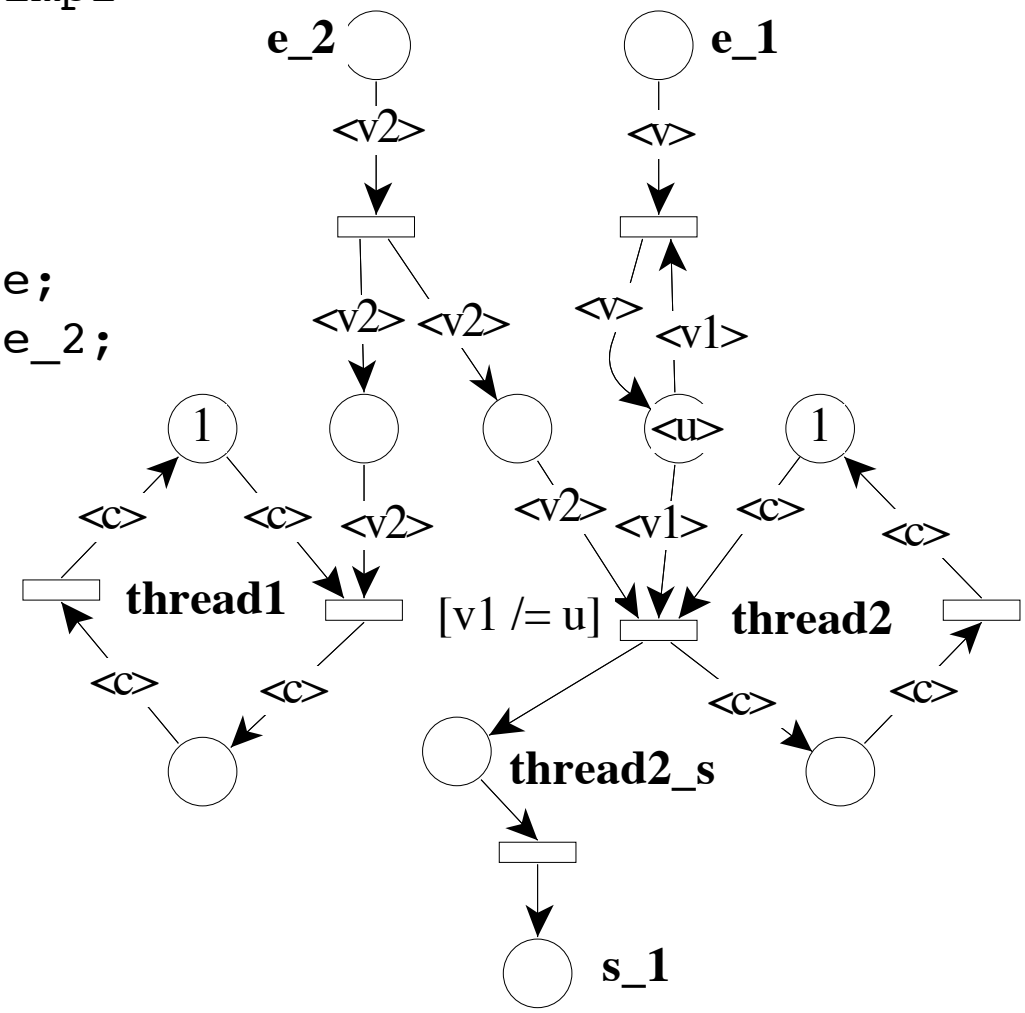
- we model the execution flows
  - circulation of data components through the architecture
  - execution of subprogram call sequences
- places model component features
- transitions model the components themselves and the AADL connections

```
process a_process
features
  e_1 : in event data port;
  e_2 : in data port;
  s_1 : out data port;
end a_process;
```



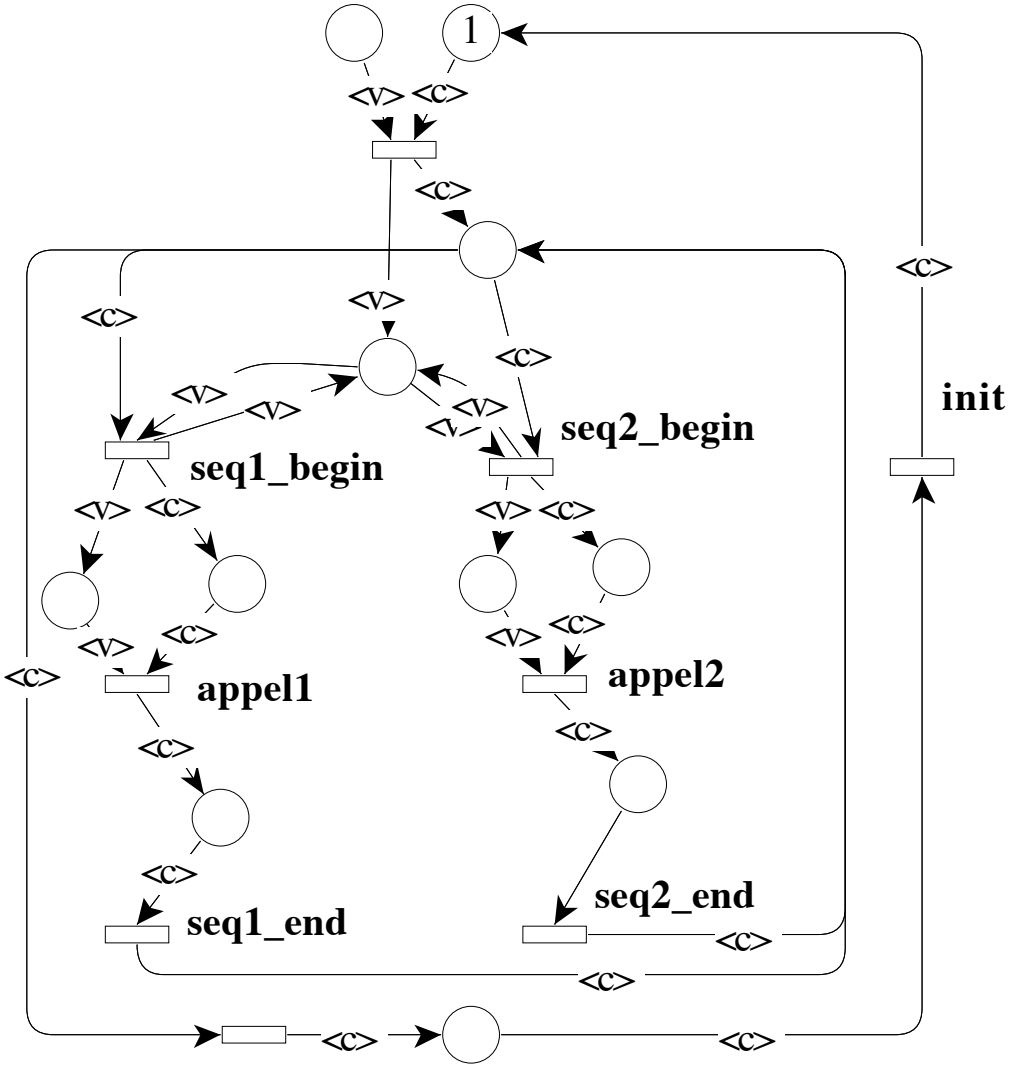
Modeling Subcomponents

```
process implementation a_process.impl
subcomponents
  thread1 : thread thread_a;
  thread2 : thread thread_b;
connections
  event data port e_2 -> thread1.e;
  event data port e_2 -> thread2.e_2;
  data port e_1 -> thread2.e_1;
  data port thread2.s_1 -> s_1;
end a_process.impl;
```

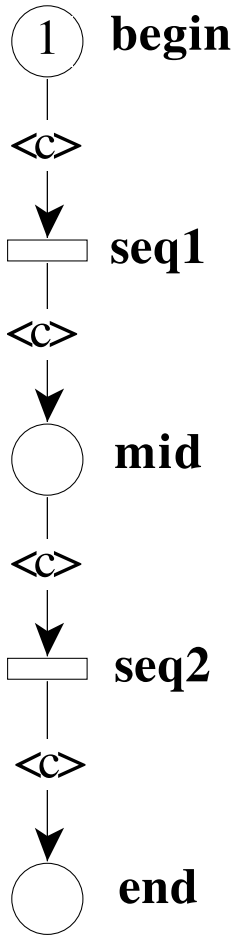


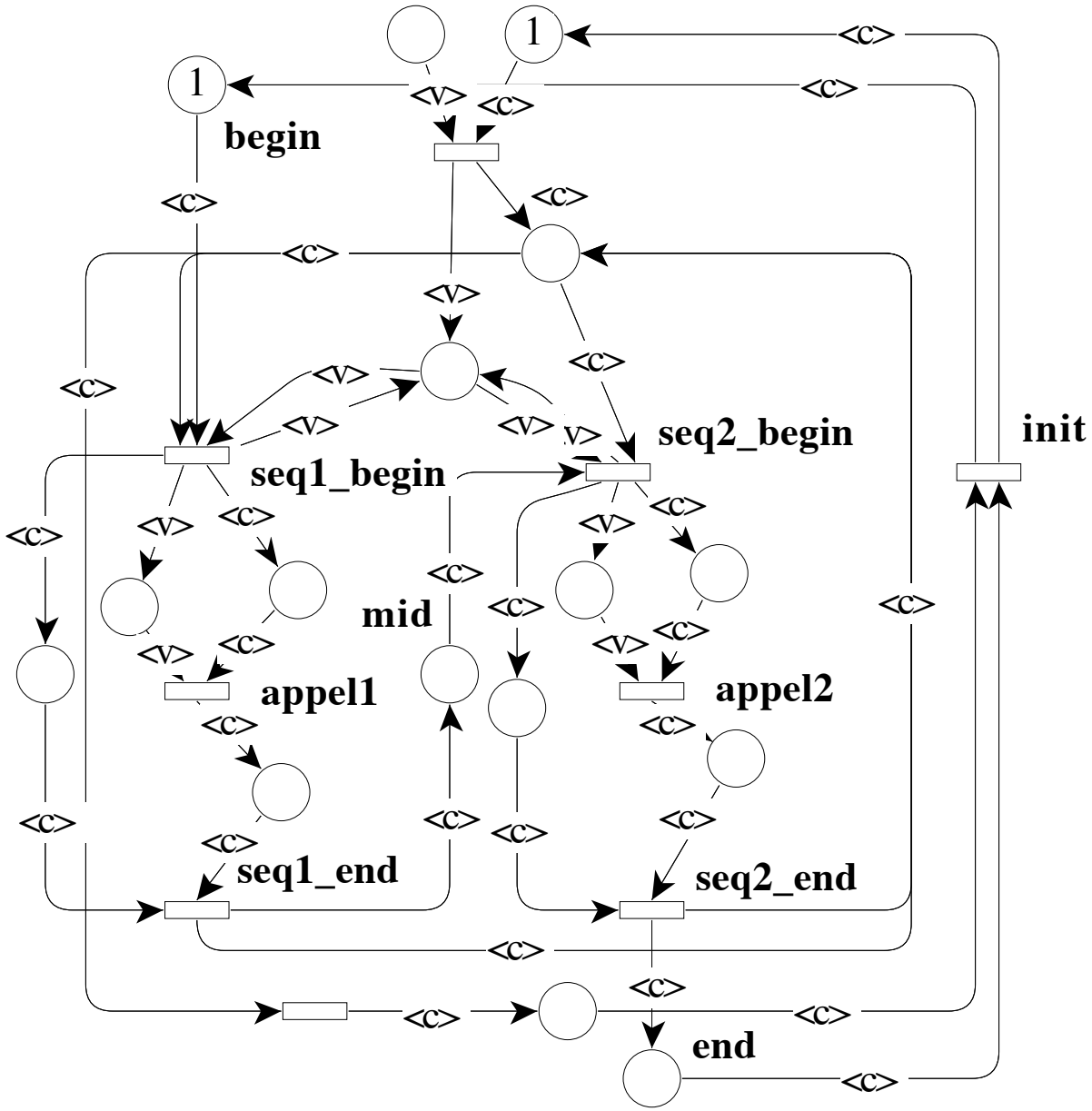
Describing Call Sequences

```
thread implementation thread_a.i
calls
  seq1 :
   {appel1 : subprogram sp1;};
  seq2 :
   {appel2 : subprogram sp2;};
connections
  parameter e_1 -> appel1.e_1;
  parameter e_1 -> appel2.e_1;
end thread_a.i;
```



Describing Behaviors





Conclusion

- Petri nets can be used to perform model checking on the AADL architecture
- not suited for time analysis
  - use schedulability analysis tools (Cheddar, ADeS?)

- verify execution flows
  - no deadlocks
  - no undefined values
- we reproduce the same runtime specifications as for code generation
  - relevant, since source code implementations do not control our AADL runtime

- the mapping between AADL and Petri nets will be described in my Ph.D. dissertation