Generating Petri Nets from AADL descriptions

Thomas Vergnaud <thomas.vergnaud@enst.fr>
Designing a Distribution Runtime for the AADL

- 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
• 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

- 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

• 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 guarantee 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;
initial_state
compute
listen
```

AADL meeting Thomas Vergnaud
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

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

1
begin

seq1

mid

seq2

end

1
begin

seq1_begin

seq2_begin

appel1

appel2

seq1_end

mid

seq2_end

end
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