AADS+: AADL Simulation including the Behavioral Annex

Fifth IEEE International workshop UML and AADL
24th March 2010, Oxford, UK

Roberto Varona Gómez
Eugenio Villar
{roberto, evillar}@teisa.unican.es
University of Cantabria, Santander, Spain
This work has been partially supported by the Spanish Ministry of Industry, Tourism and Trade through the ITEA 05015 SPICES Project and the TEC2008-04107 project.
Index

- State of the Art
- AADS
- Translation of the behavioral annex
- Case Study
- Conclusions
- References
State of the Art

Several authors have considered the behavioral annex in their research on AADL:

- P. Dissaux et al. [9] present a proposal for a behavioral annex to the AADL standard. They explain how to implement the behavioral annex with the Stood tool, a graphical AADL editor that can import and export AADL textual specifications.
- R. Bedin et al. [10] evaluate the behavioral annex through a flight software design in the ArchiDyn project. They require new synchronization primitives for AADL runtime and support using edition and analysis tools for the behavioral annex.
- J. P. Bodeveix et al. [11] propose an AADL behavioral annex and a technique to perform compositional real-time verification of AADL models through the use of a method which translates environmental constraints into behavior.
- B. Berthomieu et al. describe in [12] a formal verification tool chain for AADL with its behavioral annex available in the Topcased environment. They translate the AADL model to Fiacre and verify the behavior with a Time Petri Net Analyzer (Tina).
State of the Art

- C. Ponsard et al. explore in [13] the interplay of requirements and architecture in a model-based perspective by defining a mapping and a constructive process taking into account specificities of embedded systems, especially the importance of non functional requirements. To generate the behavioral part of a system they first generate a finite state machine and then an AADL mode-transition.

- To allow simulation M. Yassin Chkouri et al. propose in [14] a translation from AADL models into BIP models. They take into account behavior specifications allowing state variables, initialization, states and transitions sections to be defined and translating them into BIP.

- DUALLY [15] is an automated framework that allows architectural language interoperability through automated model transformation techniques. I. Malavolta et al. analyze the feasibility of integrating AADL and OSATE in DUALLY. They map AADL behavioral annex sections of states, composite states and transitions.
State of the Art

- After analyzing the state of the art, a behavioral annex to the AADL standard appears to be necessary, which could be included in an AADL model in order to express the behavior of the components.
- However, no approach uses SystemC [16], which is the recognized standard for modeling HW/SW platforms, with its great potential for integration of processors, buses, memories and specific platform HW. Our solution makes HW/SW co-design easier because of the use of SystemC.
- SCoPE [17-18] is a C++ library that extends the standard language SystemC without modifying it. It simulates C/C++ SW code based on two different operating system interfaces (POSIX [19-20] and MicroC/OS). Moreover, it co-simulates these pieces of SW code with HW described in SystemC.
- We have improved AADS to take into account the most important issues of the AADL behavioral annex (states, transitions, sending and receiving messages, etc.). AADS+ supports AADL behavioral annex simulation in SystemC, thus enabling the HW platform to be modeled and permitting HW/SW codesign. The AADL model is based on POSIX, so it supports many different RTOSs.
AADS

- AADS is written in **Java** and it was developed as a **plug-in** of **Eclipse** for **Windows**.
- AADL enables the specification of both the architecture and functionality of an embedded real-time system. **AADS translates both**, it parses the AADL model so the **functionality** is translated to an equivalent **POSIX** model and the **architecture** is represented in **XML**.
AADS

ECLIPSE

OSATE

Graphical editor

TOPCASED

Textual editor

AADL Model

Refinement

SCoPE

Configuration parameters

Simulation

Performance analysis

AADS+: AADL Simulation including the Behavioral Annex

Roberto Varona Gómez, Eugenio Villar
Translation of the behavioral annex

The transition system (an extended automaton) using optional sections:

- **State variables.** The state variables section declares typed identifiers. Types are data classifiers of the AADL model. AADS+ translates these state variables declaring variables with their corresponding type in the C++ source code of the thread or subprogram itself.

- **Initialization.** The state variables must be initialized in the initialization section using a sequence of assignments. AADS+ translates this initialization initializing the variables with their corresponding value where they were declared.

- **States.** The states section declares automaton states which can be qualified as initial, complete, return, urgent or composite. AADS+ uses this section to know which states have been defined.

- **Transitions.** The transitions section defines system transitions from a source state to a destination state. The transition can be guarded with events or Boolean conditions. An action part can be attached to the transition. It can perform subprogram calls, message sending or assignments. AADS+ translates the transitions section into switch and case statements to transit from one state to another. It starts in the initial state and moves to the next state when the guard of the transition is true. So the guard of the transition translated by AADS+ acts as a condition to execute the sentence/s of the state and to change the state. Sentence/s is the action of the transition translated by AADS+.
Translation of the behavioral annex

Depending on the content of the guard and the action of the transition, AADS+ translates them into the corresponding sentences of source code:

- **Sending / receiving messages.** Messages are sent / received through event or event data ports. If $p$ is an input port: $p?$ de-queues an event port variable, $p?$x de-queues a datum on an event data port in the variable x. If $p$ is an output port: $p!$ calls Raise_Event on an event port, $p!d$ writes data $d$ in the event data port and calls Raise_Event.

In the first case the guard of a transition is $p1?$x (where $p1$ is an in event data port) and the action of that transition is $p2!(x+1)$ (where $p2$ is an out event data port). AADS+ translates this case, checking whether a variable arrives at the POSIX message queue associated with the port $p1$. Then the variable is sent through the POSIX message queue associated with the port $p2$, in this case after adding 1 to it.

In the second case the guard of a transition is $p1?$ (where $p1$ is an in event port) and the action of that transition is $p2!$ (where $p2$ is an out event port). AADS+ translates this case, checking whether the corresponding POSIX signal associated with the port $p1$ has been received. Then the corresponding POSIX signal associated with the port $p2$ is sent.
Translation of the behavioral annex

**Subprograms.** A behavior expressed by the annex can be attached to a subprogram implementation. The behavior can refer to the subprogram parameters and to variables. The automaton specifying the subprogram implementation has one or more return *states* indicating the return to the caller. While the AADL control flows define the call sequences produced by a subprogram, the annex enables the expression of dependencies between the control flows and *state variables* or parameters. A subprogram specification can express other calls or notification of events.

In the first case the *guard* of a *transition* is \( p1? \) (\( p1 \) is an *in event port*) and the *action* of that *transition* is \( \text{subp!} \) (*subp* is a subprogram). AADS+ translates this case checking whether the corresponding POSIX signal associated with the *port* *p1* has been received. If the signal has been received then the corresponding previously defined subprogram is called.

Parameters can be passed to called subprograms. The *action* of that *transition* could be \( \text{subp!}(5\rightarrow x,2\rightarrow y) \) where \( x \) and \( y \) are two *in* parameters of the subprogram *subp*. Then AADS+ translates it into a call to the subprogram with those two parameters as *subp*(5,2).

Using the AADL behavioral annex, it is possible to indicate in the *action* of a *transition* that the *out* parameter of a subprogram is the *in* parameter modified in some way. It could be \( \text{po!}(pi+1) \), where *po* is the *out* parameter and *pi* the *in* parameter. AADS+ translates it creating the source code in the subprogram that sums one to the *in* parameter and assigns the result to the *out* parameter.

If *guard* of a *transition* is *on pi* (*pi* is an *in parameter*) and the *action* of that *transition* is a call to a standard function like \( \text{std::cout!} \). To translate this *transition* AADS+ generates the C++ source code that checks whether the *in* parameter is true and, if it is, calls the standard function *cout*.

AADDS+: AADL Simulation including the Behavioral Annex

Roberto Varona Gómez, Eugenio Villar
Translation of the behavioral annex

- **Control structures.** Control structures support conditional execution of alternative actions (*if, else, end if*), conditional repetition of actions (*while*), and application of actions over all elements of a data component array, *port* queue content, or integer range (*for*). The *For* structure represents an ordered iteration over all elements. Within the *for* structure the element can be referenced by *element_variable_identifier*, which acts as a local variable with the name scope of *for* structure.

  In the case that the *action* of a *transition* contains a conditional structure of the type: *
  
  ```
  if (logical value expression) behavior_actions [else behavior_actions] end
  if *
  ```
  
  AADS+ translates it producing the source code with the analogous *if else* structure in C++, adapting the differences between them.

  The same can be said about *for* and *while* structures of the type: *
  
  ```
  for (element variable identifier in values) {behavior_actions}
  ```
  
  and *
  
  ```
  while (logical value expression) {behavior_actions}
  ```
  
  AADS+ translates them producing the source code with the analogous *for and while* structure in C++, adapting the differences between them.

- **Arrays.** To declare collections of data which are considered to be ordered the notion of *multiplicity* is used. AADS+ translates *multiplicity* into a C++ *array of data*. The type of the *array* is the same in both AADL and C++.
Case Study

- The proposed method implemented in AADS+ has been tested in a typical case study, the cruise control, to assure the feasibility of the translation. **Cruise control** is a system that automatically controls the velocity of a motor vehicle. The driver sets a speed and the system will take over the throttle to maintain it.

- The use of the AADL behavioral annex with AADS+ has been validated through the refinement of the original cruise control design. As the original model was developed **without** using the behavioral annex, the model lacked relevant behavioral information. The annex overcomes these problems and enables the development of a more detailed architecture.
AADDS+: AADL Simulation including the Behavioral Annex

Roberto Varona Gómez, Eugenio Villar
Case Study

- The figure shows an AADL model with its behavioral annex of a cruise control system, taken from the collection of AADL examples in the OSATE release, but modified to add some subcomponents. The system component contains two processors, two memories and two devices connected by a bus, and two SW subsystems. Each of the subsystems is bound to a separate processor and to a separate memory. Threads communicate via data ports, event ports and event data ports. Some data access connections can be seen too. There are some subprograms within threads and within data subcomponents and the call sequences (local and remote) between them are shown. The parameter connections between subprograms are shown too. One subsystem has two processes, one with four threads and the other with one. The other subsystem contains one process, with two threads.
Case Study

- The files produced by AADS+ are compiled with SCoPE to simulate the model. The results obtained in the simulation are used to refine the model of the cruise control as needed.

- Example of the translation performed by AADS+ of the behavior specification of a thread: Messages are sent and received through event data ports. In this case the guard of a transition is $\text{Refspd}_\text{Mph}?x$ and the action of that transition is $\text{Filrefspd}_\text{Mph}!(x+1)$ ($\text{Refspd}_\text{Mph}$ / $\text{Filrefspd}_\text{Mph}$ are in / out event data ports). AADS+ translates it checking whether a variable arrives at the POSIX message queue associated with the port $\text{Refspd}_\text{Mph}$. Then the variable is sent through the POSIX message queue associated with the port $\text{Filrefspd}_\text{Mph}$, after adding 1 to it.
Case Study

AADL

thread Cruise1
features...
  Refspd_Mph: in event data port Float_Type {
    AADL_Properties::Queue_Size => 200;
    AADL_Properties::Queue_Processing_Protocol => FIFO;
  };
  ...
  Filrefspd_Mph: out event data port Float_Type;
end Cruise1;

thread implementation Cruise1.Simulink
properties...
annex behavior specification (*)
state variables
  CRefspd_Mph: behavior::integer;
initial
  CRefspd_Mph := 0;
states
  0: initial state;
  1: complete state;
transitions
  0 -> [Refspd_Mph? (x)] -> 1 (Filrefspd_Mph!(x+1));
**};
end Cruise1.Simulink;

POSIX / C++

float mens;
int len;
unsigned int prio;
len = mq_receive( Refspd_queue, (char *)&mens,
                 sizeof(mens), &prio );
if( len > 0 ) {
  cout << "Cruise1 receives message from "+
       "Refspd containing " << mens << endl;
  NRFspd_Mph = mens;
  CRefspd_Mph = 1;
}
...
int stateCruise1 = 0;
...
switch( stateCruise1 ) {
  case 0:
    if( CRefspd_Mph | ( 
        Float mens = NRFspd_Mph+1;
        cout << "Cruise1 sends a message "+
            "containing " << mens << endl;
        mq_send( Filrefspd_queue, 
            (const char *)&mens, sizeof(mens), 1);
        CRefspd_Mph = 0;
        stateCruise1 = 1;
    )
      break;
  case 1:
    break;
  default:
    break;
}
Case Study

- *Subprograms* with their behavior specifications have been added to the AADL model of the cruise control to obtain the desired system performance. For example, to detect if a *button* has been *pushed* by the driver the corresponding *behavior* was added to a *subprogram* in *Button_panel thread* and refined through simulation.

- When the *driver* activates the cruise control, an *event* is sent to the *Refspd thread* that *sends* another *event* to the *Instrumentpanel thread* to show the activation; this behavior has been implemented in the *thread Refspd*.

- The correct operation of the behavior specification created to know whether the *Drivermodelogic is activated or disactivated* was refined by simulating the model.
Case Study

- Refinement of the original cruise control model with behavior specifications does not require a large number of AADL code lines, AADS+ does not produce so many C++ code lines as one might fear and the gain in expressiveness of the model’s behavior is great. Furthermore, the cost in terms of use of CPU, core energy/power, bus access time, etc is slight.

<table>
<thead>
<tr>
<th></th>
<th>AADL model without behavioral annex</th>
<th>AADL model refined with behavioral annex</th>
</tr>
</thead>
<tbody>
<tr>
<td>AADL lines</td>
<td>560</td>
<td>682</td>
</tr>
<tr>
<td>C++ &amp; XML lines</td>
<td>1954</td>
<td>2207</td>
</tr>
<tr>
<td>Number of thread switches</td>
<td>6936</td>
<td>6567</td>
</tr>
<tr>
<td>Running time</td>
<td>3,927344674 s</td>
<td>3,942027992 s</td>
</tr>
<tr>
<td>Use of CPU</td>
<td>98.1836 %</td>
<td>98.5507 %</td>
</tr>
<tr>
<td>Instructions executed</td>
<td>330919376</td>
<td>332182295</td>
</tr>
<tr>
<td>Instruction cache misses</td>
<td>15340</td>
<td>16900</td>
</tr>
<tr>
<td>Core Energy</td>
<td>66.18386e+07 nJ</td>
<td>66.43646e+07 nJ</td>
</tr>
<tr>
<td>Core Power</td>
<td>165.4594 mW</td>
<td>166.0914 mW</td>
</tr>
<tr>
<td>Instruction cache energy</td>
<td>9.93372e+08 nJ</td>
<td>9.97223e+08 nJ</td>
</tr>
<tr>
<td>Instruction cache power</td>
<td>248.343 mW</td>
<td>249.3058 mW</td>
</tr>
<tr>
<td>Bus access time</td>
<td>5830500 ns</td>
<td>6636280 ns</td>
</tr>
<tr>
<td>Idle time</td>
<td>4057597179 ns</td>
<td>4046782848 ns</td>
</tr>
<tr>
<td>Number of interrupts</td>
<td>4855</td>
<td>4575</td>
</tr>
</tbody>
</table>
Conclusions

- The paper presents the simulation of the AADL behavioral annex using the AADS+ simulation tool. AADS+ supports the refinement of AADL models, including the behavioral annex, after translating those models, through performance analysis done with SCoPE.

- Future work includes incorporation of AADS+ features that appear in V2.0 of the AADL standard. Furthermore, the source code produced by AADS+ for the software components will be made compatible with the ASSERT Ravenscar Computational Model (RCM).
References

References

Thanks for your attention.