COTRE as an AADL profile

Pierre GAUFILLET & Patrick FARAIL
AIRBUS FRANCE
pierre.gaufillet@airbus.com / patrick.farail@airbus.com
Tel. : +33 (0)5.61.18.84.85 / +33 (0)5.61.93.66.28
COTRE overview 1/2

- Funded by the French research department (total 1.9M€, 230 m.m), from 2002 to 2004

- Goal: Real Time architecture verification (mainly from the behavioral point of view)

- Exploration project aiming to develop a demonstration tool

- Partners: AIRBUS, TNI-Valiosys, IRIT, LAAS, ONERA-DTIM, ENSTB
COTRE overview 2/2

Designer

HOOD mapping
UML Mapping

Model of Library
O.S., Hardware...

COTRE Description Language
AADL SAE Standard compatible

Architecture +
Behaviour

Properties
C1

Output/ Diagnosis

Extraction/Translation

FC1
FC2

Simulation/ Verification/ Evaluation/ Test

Schedulability
Timed and Untimed Properties
Performance Analysis
UCOTRE

- User level COTRE language, opposed to VCOTRE, Verification COTRE language. The 2 languages seems mature enough today to be merged.

- Based initially on the HOOD IV and HRT-HOOD concepts, and then on AADL.

- The main restriction to AADL is the lack of ports: every communications are modeled by subprogram calls. Ports should be added for multi-process applications.
AADL extension mechanisms

- Property sets:

PROPERTY SET my_ext IS
  int : TYPE INTEGER -32767..32768;
  duration : TYPE UNITS (s, j => s * 86400,
                     ms => s * 0.001,
                     us => s * 0.000001);
  rate : float => 0.0 APPLIES TO (SUBPROGRAM);
END my_ext;

- Annexes:

ANNEX <name> IS
  <free syntax>
END ANNEX <name>;
Deadlock system

SYSTEM deadlock_verification
END deadlock_verification;

SYSTEM IMPLEMENTATION deadlock_verification.default
SUBCOMPONENTS
  dp : PROCESS Partition.A;

ANNEX cotre.guarantees IS
IS ALIVE;
END ANNEX cotre.guarantees;
END deadlock_verification.default;

PROCESS Partition
END Partition;

PROCESS IMPLEMENTATION Partition.A
SUBCOMPONENTS
  t_1 : THREAD t.t1(sem1 => sem_1, sem2 => sem_2);
  t_2 : THREAD t.t2(sem1 => sem_2, sem2 => sem_1);
  sem_1 : DATA semaphore.default;
  sem_2 : DATA semaphore.default;
END Partition.A;

sys_deadlock_instance: SYSTEM ex_deadlock.default { };
# Contracts

**ANNEX**

cotre.guarantees | cotre.assumes IS

((<assertion> | <behavioral equivalence> | <raw formula>);)+

**END ANNEX**
cotre.guarantees | cotre.assumes;

<table>
<thead>
<tr>
<th>Assertion</th>
<th>Formal description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>potentially reset</td>
<td>AG EF init</td>
<td>From any state, the component may go back to its initial state.</td>
</tr>
<tr>
<td>unavoidably reset</td>
<td>AG AF init</td>
<td>From any state, the component must go back to its initial state.</td>
</tr>
<tr>
<td>is alive</td>
<td>AG EF EX&lt;sub&gt;c&lt;/sub&gt;true</td>
<td>Some actions have always to be possible in the future. Applied to a root component, this assertion implies that there is no deadlock.</td>
</tr>
<tr>
<td>no livelock</td>
<td>AG AF EX&lt;sub&gt;c&lt;/sub&gt;true</td>
<td>The component must not stay forever idle.</td>
</tr>
<tr>
<td>invariant&lt;exp&gt;</td>
<td>AG &lt;exp&gt;</td>
<td>&lt;exp&gt; has always to be true.</td>
</tr>
<tr>
<td>&lt;exp1&gt; leads to &lt;exp2&gt;</td>
<td>AG(e1=&gt;AF&lt;sub&gt;d&lt;/sub&gt;e2)</td>
<td>The occurrence of &lt;exp1&gt; always implies the occurrence of &lt;exp2&gt; in less time than &lt;exp3&gt;.</td>
</tr>
<tr>
<td>[within &lt;exp3&gt;]</td>
<td>AG(e1=&gt;EF&lt;sub&gt;d&lt;/sub&gt;e2)</td>
<td>The occurrence of &lt;exp1&gt; may imply the occurrence of &lt;exp2&gt; in less time than &lt;exp3&gt;.</td>
</tr>
<tr>
<td>reachable &lt;exp1&gt;</td>
<td>AG(e1=&gt;EF&lt;sub&gt;d&lt;/sub&gt;e2)</td>
<td>The occurrence of &lt;exp1&gt; may imply the occurrence of &lt;exp2&gt; in less time than &lt;exp3&gt;.</td>
</tr>
<tr>
<td>[from &lt;exp2&gt;] [within &lt;exp3&gt;]</td>
<td>AG(¬EU(¬e2, e1))</td>
<td>&lt;exp2&gt; always occurs after &lt;exp1&gt;.</td>
</tr>
</tbody>
</table>
Modeling threads

THREAD t
  REQUIRES
    sem1 : DATA ACCESS semaphore;
    sem2 : DATA ACCESS semaphore;
  END t;

THREAD IMPLEMENTATION t.t1
  PROPERTIES
    Period => 13.96ms;
    cotre::Priority => 1;
    cotre::Phase => 0.0ms;
    Dispatch_Protocol => Periodic;

ANNEX cotre.behavior IS
  STATES
    s0, s1, s2, s3, s4, s5, s6, s7, s8 : STATE;
    s0 : INITIAL STATE;
  TRANSITIONS
    s0 -[]-> s1 { PERIODIC_WAIT };
    s1 -[]-> s2 { COMPUTATION(1.9ms, 1.9ms) };
    s2 -[ sem1.wait ! (-1.0ms) ]-> s3;
    s3 -[]-> s4 { COMPUTATION(0.1ms, 0.1ms) };
    s4 -[ sem2.wait ! (-1.0ms) ]-> s5;
    s5 -[]-> s6 { COMPUTATION(2.5ms, 2.5ms) };
    s6 -[ sem2.release ! ]-> s7;
    s7 -[]-> s8 { COMPUTATION(1.5ms, 1.5ms) };
    s8 -[ sem1.release ! ]-> s0;
  END ANNEX cotre.behavior;
END t.t1;
Behavior 1/3

- Applies to threads and subprograms
- Mealy machines & Timed Transitions Systems

VARS
  <variable> : <type>;

INITS
  <variable> := <value>;

STATES
  <state name>, <state name>* : STATE;
  <state name> : INITIAL STATE;

TRANSITIONS
  ...

EXCEPTIONS
  ...
Behavior 2/3

- **Transitions**
  
  \[ (<label>:\)* <origin state> -[<clearing condition>]\rightarrow <arrival state> \{<actions>\} ; \]

- **Exceptions**
  
  \[ <sensibility cond.> -[ <clearing cond.> BEFORE <end of sensibility cond.> ]\rightarrow <final state> \{ <actions> \} ; \]

- **Conditions**
  
  WHEN <boolean condition> \rightarrow <synchronization event>

<table>
<thead>
<tr>
<th>Synchronization events</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;subprogram name&gt; ! ((&lt;parameters&gt;))</td>
<td>Calls the named subprogram with the required parameters. The subprogram is identified by using the dotted notation &lt;object&gt;.&lt;subprogram&gt;. Parameters are separated by commas.</td>
</tr>
<tr>
<td>CALLED ?</td>
<td>The subprogram being described is called.</td>
</tr>
<tr>
<td>RESUME((&lt;parameters&gt;))</td>
<td>Give back the control to the caller (but the subprogram being described can go on running).</td>
</tr>
</tbody>
</table>
# Behavior 3/3

## Actions

<table>
<thead>
<tr>
<th>Actions</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COMPUTATION</strong>(&lt;max_duration or range&gt;)</td>
<td>Consumes a CPU time smaller then &lt;max_duration&gt; or bounded by &lt;range&gt;.</td>
</tr>
<tr>
<td><strong>DELAY</strong>(&lt;max_duration or range&gt;)</td>
<td>The execution is deferred for a time smaller than &lt;max_duration&gt; or bounded by &lt;range&gt;.</td>
</tr>
<tr>
<td><strong>PERIODIC_WAIT</strong></td>
<td>Delays the execution of a periodic thread until the beginning of its next period. For an aperiodic thread, it is equivalent to <strong>SKIP</strong>.</td>
</tr>
<tr>
<td><strong>SKIP</strong></td>
<td>Does nothing.</td>
</tr>
<tr>
<td>&lt;l_exp&gt; := &lt;exp&gt;</td>
<td>Modifies the value of the variable &lt;l_exp&gt;.</td>
</tr>
</tbody>
</table>
Modeling semaphore

DATA semaphore
   PROVIDES
      wait: SUBPROGRAM;
      release: SUBPROGRAM;
   PROPERTIES
      cotre::Protected => TRUE;
END semaphore;

DATA IMPLEMENTATION semaphore.default
   PROPERTIES
      sem_p::Max_tokens => 1;
END semaphore.default;

ANNEX cotre.behavior IS
   VARS
      tokens : INTEGER 0..+infinity;
   INITS
      tokens := sem_p::Max_tokens;
   SUBPROGRAM wait
      STATES
         s0, s1 : STATE;
         s0 : INITIAL STATE;
      TRANSITIONS
         s0 -[ WHEN tokens > 0 => CALLED ? ]-> s1 { tokens := tokens - 1 };
         s1 -[ RESUME ]-> s0;
   END ANNEX cotre.behavior;

END semaphore.default;
Types

- Basic types: AADL data, Boolean, integer, real, ranges, with or without units.
- Data definition is required for behavioral verification.

```plaintext
DATA IMPLEMENTATION <name>
ANNEX cotre.type IS
  <field 1> : <type 1>;
  …
  <field n> : <type n>;
END ANNEX cotre.type;
END <name>;
```

- Or:

```plaintext
DATA IMPLEMENTATION <name>
ANNEX cotre.type IS
  <type>;
END ANNEX cotre.type;
END <name>;
```
## COTRE Property Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Applies to</th>
<th>Default value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>cotre::Ceiling_priority</td>
<td>integer</td>
<td>DATA</td>
<td>-</td>
<td>For data protected using PCP.</td>
</tr>
<tr>
<td>cotre::Description</td>
<td>string</td>
<td>any component and subcomponent</td>
<td>-</td>
<td>Informal comments, traceability informations, ...</td>
</tr>
<tr>
<td>cotre::Min_Time</td>
<td>time</td>
<td>THREAD</td>
<td>-</td>
<td>Minimum time between 2 sporadic thread activations.</td>
</tr>
<tr>
<td>cotre::Phase</td>
<td>time</td>
<td>THREAD</td>
<td>0.0s</td>
<td>Offset from the beginning of the period for periodic threads.</td>
</tr>
<tr>
<td>cotre::Priority</td>
<td>integer</td>
<td>THREAD</td>
<td>-</td>
<td>Base priority of the thread (semantic is scheduling policy dependent).</td>
</tr>
<tr>
<td>cotre::Protected</td>
<td>boolean</td>
<td>SUBPROGRAM</td>
<td>false</td>
<td>true if the subprograms of the component are executed exclusively.</td>
</tr>
<tr>
<td>cotre::Reentrant</td>
<td>boolean</td>
<td>SUBPROGRAM</td>
<td>false</td>
<td>true if the subprogram is reentrant.</td>
</tr>
</tbody>
</table>
Conclusion

- Main differences with AADL are:
  - The lack of ports
  - Behaviors
  - Types
  - Contracts

- The AADL core used have to be upgraded.
- Support for multi-application models has to be added (using ports).
- UCOTRE/VCOTRE merge still to be done.
- Some tuning to do at the property set level and for variables (using data instead).
- A partial implementation of the method exists. An industrial-strength open source tool supporting COTRE is planned in the forthcoming TOPCASED project.