Two formal semantics of a Subset of AADL

Z. Yang\textsuperscript{1}, K. Hu\textsuperscript{1}, J.-P. Bodeveix\textsuperscript{2}, L. Pi\textsuperscript{2}, D. Ma\textsuperscript{1}, J.-P. Talpin\textsuperscript{3}

\textsuperscript{1}BUAA, Beijing, China
\textsuperscript{2}IRIT - Université Paul Sabatier, Toulouse, France
\textsuperscript{3}INRIA-Rennes, Campus de Beaulieu, Rennes, France

April 2011
Plan

1. Introduction

2. Formalisms
   - TASM
   - Semantics domain: TTS
   - Coq

3. Verification

4. Conclusion
Context

Principles

- **Goal:** verification of AADL models
- **Method:** translation to an analysable formal language (TASM here)
- **Problem:** correctness of the transformation
- **Verification:** Semantics-preserving transformation
Semantics-preserving transformation

- AADL model
- Formal model (TASM)
- Behavioral model

Transformation: model transformation
Semantics: semantics
Bisimulation: bisimulation
Formalisms

- AADL: synchronous subset (periodic threads, immediate/delayed communications)
- Formal analysable language: TASM (Timed abstract state machine)
- Semantics domain: TTS (Timed Transition systems)
- Meta-language: Coq (interactive proofs)
Plan

1. Introduction

2. Formalisms
   - TASM
   - Semantics domain: TTS
   - Coq

3. Verification

4. Conclusion
TASM [Lundqvist, MIT]: Extension of abstract state machines

- system: set of concurrent machines
- machine: set of transitions updating local or shared variables
- time: duration associated to a transition
- resources: consumed during transition execution
TASM (2)

computed: bool
write: bool
read: bool

producer

R0:

{  
t := 0;
    if write = true then
        write := false; read := true;
}

R1:

{  
t := 5; mem := 50;
    if read = true then
        read := false; computed := true;
}

R2:

{  
t := 0;
    else then skip
}
Semantics

- One machine: transition system
  - select a rule with satisfiable guard
  - compute update set (next value of updated variables)
  - wait for transition duration, consume (additively) resources
  - update environment

- Composition of machines:
  - Asynchronous if zero-time transitions
  - Synchronous if non-zero time transitions
A TTS over a set of events $\Sigma$ is a nuple

$$\langle Q, Q^0, P, \rightarrow, \models \rangle$$

where

- $Q$ is a set of states,
- $Q^0 \subseteq Q$ is the set of initial states
- $P$ is a set of observables predicates
- $\_ \rightarrow \_ \subseteq Q \times (T \cup \Sigma) \times Q$ is the timed transition relation
- $\models \subseteq Q \times P$ is a satisfaction relation.
Simulation relation

∀c, c ∈ Q^0_c ⇒ ∃a, R(c, a) ∧ a ∈ Q^0_a

∀c_1 c_2 a_1 e,

\[ c_1 \xrightarrow{e} c_2 \land R(c_1, a_1) \]

⇒

∃a_2, a_1 \xrightarrow{e} a_2 \land R(c_2, a_2)
### Principle

\[ \forall i, \text{TTS\_TASM}_i \sim \text{TTS\_AADL}_i \]

\[ \prod_{i=1}^{n} \text{TTS\_TASM}_i \sim \prod_{i=1}^{n} \text{TTS\_AADL}_i \]

### Synchronous product of TTS

\[ \langle Q_1, Q_0^1, P_1, \xrightarrow{1}, \models_1 \rangle \otimes \langle Q_2, Q_0^2, P_2, \xrightarrow{2}, \models_2 \rangle = \]

\[ \langle Q_1 \times Q_2, Q_0^1 \times Q_0^2, P_1 \cup P_2, \xrightarrow{1}, \models_1 \cup \models_2 \rangle \]

where

\[ q_1 \xrightarrow{e} q_1' \quad q_2 \xrightarrow{e} q_2' \]

\[ (q_1, q_2) \xrightarrow{e} (q_1', q_2') \]
Coq and proof assistants

Several languages (maybe unified)
- typed functional language (definition of functions)
- rich type system (functions over types, dependent types, ...)
- assertional language (definition of properties)
- proof building language (assisted proofs)
- proof tactic definition language (proof automation)
Use of Coq

- Definition of TTS, product, simulation
- Definition of TASM and AADL abstract syntax
- Definition of TASM and AADL semantics (as a product of TTS)
- Definition of AADL to TASM translation.
- Proof of bisimulation.

Feasability validated on a small subset of AADL.
Plan

1. Introduction
2. Formalisms
   - TASM
   - Semantics domain: TTS
   - Coq
3. Verification
4. Conclusion
Abstract syntax of an AADL subset (Coq)

Record Thread: Type := {  
  WCET: Duration;  
  period: Duration;  
  deadline: Duration;  
  Iports: Set;  
  OPorts: Set  
}.

Record Model: Type := {  
  thId: Set;  
  threads: thId → Thread;  
  connections: ∀ th, Iports (threads th) →  
    {th: thId & Oports (threads th)};  
  connectionType: ∀ th, Iports (threads th) → bool  
}.
Abstract syntax of a TASM subset

\[
P ::= \text{Ident} ::= \text{exp} \\
| \text{skip} \\
| \text{channel!} \\
| \text{channel?} \\
| \text{if exp then } P \\
| \text{time } \text{min} .. \text{max} \triangleright P \\
| \text{time } \text{next} \triangleright P \\
| \text{resource } \text{r} (\text{min, max}) \triangleright P \\
| P \oplus P \text{ choice} \\
| P \otimes P \text{ multi assignment}
\]

\[
\text{TASM ::= } \langle \text{Env}, P \parallel ... \parallel P \rangle
\]
Variable to be updated at end of transition

Record UpdateSet: Type := mkUS { 
  date: Time;
  updated: Vars sys -> Prop;
  update: forall v, updated v -> Value sys v
}.

TASM state

Record TASMState: Type := mkTASMState { 
  currentTime: Time;
  updateSets: mmId sys -> option UpdateSet;
  currentEnv: Env (Vars sys) (Value sys)
}.
Operational Semantics of the AADL subset

Formal Semantics

```
waiting_dispatch

[ date + d <= NextPeriod ]/\delay(d)

|--\_op buffer(dest(op)) := val(op)

wait_deadline

[ date+d<=Deadline ]/\delay(d)

|--\_op,ip:imm

  buffer(ip) := val(op)

  hasCPU := false

completed

[w=0]

|--\_op val(op) := ...

execution

[d <= w]/\delay(d)/w := w-d

waiting_execution

[ date=NextPeriod &&

  inputs not in wait_dl ]

|--\_ip val(ip) := buffer(ip)

NextPeriod := NextPeriod+Period

[ hasCPU ]

|--\_ip:imm

  val(ip) := buffer(ip):

  w := WCET
```
Translation of AADL to TASM

Trans_Thread(th) =

// dispatch

{time 0}

if state(th) = \texttt{waiting\_dispatch} and
\forall \ thi \in \text{prec}(th), \ state(thi) \neq \texttt{wait\_deadline} then
state(th) := \texttt{waiting\_execution} \times
\times \ \forall ip \in \text{Iports}(th) \ \text{val}(ip) := \text{buffer}(ip)

⊕

// waiting execution

{time 0}

if state(th) = \texttt{waiting\_execution} and hasCPU(th) then
state(th) := \texttt{execution} \times
\times \ \forall ip \in \text{Iports}(th) \cup \text{Imm} \ \text{val}(ip) := \text{buffer}(ip)

⊕ ...
The main theorem (1)
The main theorem (2)

Coq statement (for one thread)

**Theorem**  Thread2MM_simu1:
\[ \forall \text{th}, \\simu \_ \_ \_ \text{A2T } (\text{ThreadPred sys th}) \]
\[ (\text{MM_TTS AADL2TASM th}) \]
\[ (\text{Thread_TTS sys th}) \]
\[ (\text{AP2TLP th}) \]
\[ (\text{P2LP th}). \]

**Theorem**  Thread2MM_simu2:
\[ \forall \text{th}, \\simu \_ \_ \_ \text{T2A } (\text{ThreadPred sys th}) \]
\[ (\text{Thread_TTS sys th}) \]
\[ (\text{MM_TTS AADL2TASM th}) \]
\[ (\text{P2LP th}) \]
\[ (\text{AP2TLP th}). \]
Plan

1 Introduction

2 Formalisms
   • TASM
   • Semantics domain: TTS
   • Coq

3 Verification

4 Conclusion
Conclusion and perspectives

**Conclusion**

- Definition of the semantics of a (small) fragment of AADL.
- Definition of the semantics of a fragment of TASM.
- AADL2TASM transformation and correctness proof.
- The proof is tedious and too dependent on AADL

**Future work**

- A higher level formalism to express the reference semantics.
- Larger coverage of AADL.
- Translation of this language to analysable languages.
- Verification of this translator.