A Journal of Software Engineering and Applications, **2012, 5, 187-192**
doi:10.4236/jsea.2012.512b036 Published Online December 2012 (http://www.scirp.org/journal/jsea)

Formal Modeling and Analysis of AADL Threads in Real Time Maude

F. Belala[1], M. Benammar[1], K. Barkaoui[2], A. Hicheur[2]

[1]Department of Computer Science, Mentouri University, Constantine, Algeria; [2]CEDRIC, Conservatoire National des Arts et Métiers, Paris, France.
Email: belalafaiza@hotmail.com, kamel.barkaoui@cnam.fr

Received 2012

ABSTRACT

This paper presents, without altering the AADL meta-model, a formal description of static and behavioral aspects of the AADL thread component. This active and concurrent applicative component of AADL poses many challenges to its formalization and analysis including instantaneous and/or delayed communications, concurrent tasks and time- dependent features, and the need to analyze correctness. This formalization, based on real-time object-oriented theories, allows not only a precise description of the semantics of threads composition with respect to their timing requirements but also makes possible the formal verification of behavioral properties.

Keywords: Architecture Description Language (ADL), AADL thread component, Real Time Maude, Model Checking.

1. Introduction

The problem of ensuring as early as possible the correctness of a software architecture (SA) is of great importance in the development life-cycle of software products. Recently, model checking, that is completely automatic, allows developers to perform exhaustive verification of the SA (as for e.g. [1]). In this paper, we will use this technique to assess whether an SA specification noted in AADL language satisfies desired architectural properties.

The architecture description language AADL (Architecture Analysis and Design Language) [2] is a formal notation for describing the architectural plan of real time embedded system at various abstraction levels. It offers the capacity to assembly both information concerning the application structure and its deployment.

However, AADL is focused on the architectural aspects of the system components and their connections, but doesn't deal directly with their behavioral aspects. Thus, the formal reasoning on AADL architecture or its formal analysis poses a set of challenges, including: 1) Modeling concurrent and distributed components of AADL architecture, 2) Modeling components communication with transmission delays, 3) Modeling time-dependent behaviors, and 4) Analyzing behavior properties with respect of AADL declared constraints.

Several works are being done in literature tempting to formalize AADL architectural description while transforming it to other formal models. These models are provided on the basis of some well known formalisms such as, timed automaton, timed Petri nets (TPN), real time process algebra (ACSR) or timed abstract state machine (TASM), that come with some convenient tools (TINA [5], CADP [6], etc.). Others more practical works are also raised; a translation from VTS (Visual Timed event Scenarios) to Time Petri Nets in [9] enables the model-checking of properties expressed in VTS over AADL models using TPN-based tools. Nevertheless, most of these contributions have focused on detecting deadlocks states or on concurrent access control to shared data, they have been interested by formalization of only some AADL concepts.

This paper shows how we can met all the cited challenges by extending the semantic framework ABAReL annex proposed in [2] and based on rewriting logic. Our extension consists in defining the behavior specification of the fundamental concurrent AADL execution unit *Thread* as a real-time rewrite theory $\mathfrak{R}_T = (\Sigma_T, E_T, L_T, R_T)$. The membership equational theory (Σ_T, E_T) describes all declared elements in the AADL thread description (*flows*, *properties*, etc.). The labeled rewrite rules (L_T, R_T), *instantaneous* and *tick* ones, describe the thread behavior according to its configurations evolution, taking into account its local states and those of its connection ports, as well as its declared timing requirements. In addition, proof of generic and crucial architectural properties (such as *safety* and *liveness*) is successfully achieved via the RT-Maude model-checker.

Real-Time Maude [4] extends the rewriting logic tool Maude to support formal specification and analysis of object based real-time systems which are suitable for modeling AADL architectural system. The high-performance RT-Maude tool provides a range of analysis techniques, including time-bounded linear temporal logic model checking**,** being used in our study.

The remainder of this paper is organized as follows. Section 2 introduces basic concepts of both Real-Time Maude language and AADL notations. In section 3, we show how we enrich ABAReL annex to formal specify, simulate and further analyze AADL architecture with several communicating threads. A case study is given in section 4 to illustrate and validate the feasibility of our approach. Finally, we conclude by a synthesis of achieved work and related prospects to its continuation.

2. Basic Concepts

2.1. RT-Maude Overview

Rewriting Logic [10] has many operational environments implementing its theoretical concepts, the most known is the Maude system [11]. The objective of this section is to present essential concepts of RT-Maude [4], an extension of Maude language appropriate to model and analyze object oriented real time systems.

A RT-Maude module implements real-time rewrite theory $\Re$ which is a quadruplet of the form [Σ, E, (L, R), (L', R')], where (Σ, E) is a membership equational logic theory, the signature Σ and conditional equations E describe system states as an algebraic data type. It must include a special sort *Time* modeling a time domain (dense or discrete). L are labels of the conditional instantaneous rewrite rules R. Each rewrite rule is noted *crl [l]: [t] → [t'] if cond* and models the possible instantaneous local transition between states of the concurrent real-time system. *[t], [t']* and *cond* are equivalence classes of algebraic terms belonging to the set $T_{\Sigma,E}(X)$. *L'* is another labels set for *tick* rewrite rules *R'* written with a particular syntax *crl [l'] : {t} => {t`} in time D if cond*. They model the advance of a *time D* in the system of state *t* if a condition *cond* is checked. *D* of sort *Time* denotes the duration of the rewrite. *{_}* is a free constructor of sort *GlobalSystem*.

The syntax of RT-Maude modules supports object-oriented concepts as objects, messages, classes, and multiple class inheritance. A real-time system state in this case is modeled by a multiset of objects and juxtaposed messages, usually called a configuration, it is a term of the built-in sort *Configuration*. Concurrent interactions between the objects are governed by rewrite rules. An object is an instance of class. It is possible to declare subclasses and profit from the heritance concept. The messages are declared using the key word *msg*. The dynamic behavior of concurrent object systems is axiomatized by specifying each of its concurrent transition patterns as a rewrite rule.

RT-Maude offers a variety of analysis tools. Among them the linear temporal logic Maude model checker used in our case study, it checks whether each behavior "up to a certain time," as explained previously, satisfies a temporal logic formula. Entries of this tool are the system behaviour specification (as RT-Maude modules) and the properties that we want to state and prove about this system. The output result is a positive answer if the property is satisfied, and one counterexample should the opposite occurs.

2.2. Overview of the AADL Language

AADL (Architecture Analysis and Design Language) is devoted to real time embedded systems description [2]. It describes software applications, their execution platforms and how components are composed and interact to form complete system architectures. It also describes component functional interfaces, component connections and how software components are allocated to hardware ones.

The abstract declaration of AADL component contains *component type* and one or more *implementations*. The component *type* declaration specifies functional interface in terms of *features*, *flow*, and *properties*. A *component implementation* specifies its internal structure in terms of *subcomponents*, *connections* between the subcomponents features, *flows* across a sequence of subcomponents, *modes* to represent operational states, and *properties*.

AADL offers *thread component* like schedulable units for the concurrent execution, *processes* to represent spaces of virtual addresses, and *systems* to support the hierarchical organization of both *threads* and *processes*. AADL also allows the *execution platform* (or hardware components) modelling in terms of *processors* (supporting the execution of threads), *memory* for the data and code storage, and *bus* to represent the physical interconnection. AADL *properties* are the key concept of this language. They describe through its own notations, some constraints on the architectural elements, components, subcomponents, interface elements, connections, etc. For each component, one can associate properties and give them values.

In order to integrate the academic research results in the industrial development process, AADL has some recent tools developed within the European project framework ASSERT (Automated proof-based System

Software Engineering for Real-Time systems) [13]. However, the AADL standard uses a restricted formal model based on operational mode and its corresponding transitions rules to express the commutation between system execution configurations representing the behaviour of a system.

We are mainly interested in this paper by giving a formal setting for the AADL component *thread,* in order to define both its structure and behaviour and also to give a mathematical semantic for its execution model.

3. The Real-Time Maude Specification of AADL Threads

Traditionally the AADL standard defines simplified automata based semantics for threads (figure 1). A thread can be stopped, inactive, or in activity. An active thread can be waiting for dispatch, computing (*Compute* state), or *blocked* on resource access. Only active threads execute their instructions. The standard gives also the possibility of specifying the implementation conditions of the threads by the declaration of some properties (*deadline, dispatch protocol, period, e*tc).

The complex task of a thread carried out at the *Compute* state (figure 1), consists in data (and/or events) receipt, computation and signals data (and/or event) send. Particularly, the hierarchical *Compute* state of this automaton is composed of others substates (figure 2). However, this execution semantics of AADL thread does not appear at the architecture description level. Only possible modes and their execution properties are declared. Our work aims to extend the behavioral annex (ABAReL for AADL Behavioral Annex based on Rewriting Logic) to define behavioral aspects of an AADL threads composition without altering the AADL meta-model.

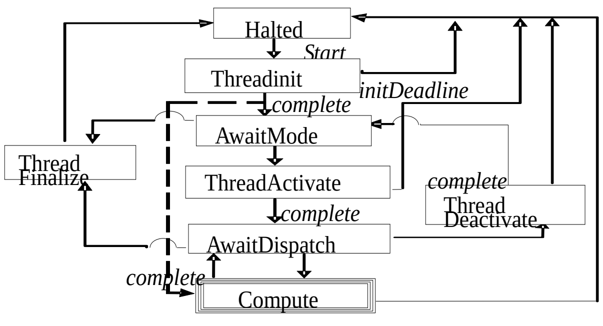


Figure 1. The state-transition model of AADL thread.

3.1. Modeling AADL Thread

We associate to an AADL thread component *T*, a formal mathematical model, represented by a Real-Time rewrite theory $R_T = (\Sigma_T, E_T, L_T, R_T)$, where (Σ_T, E_T) is a member

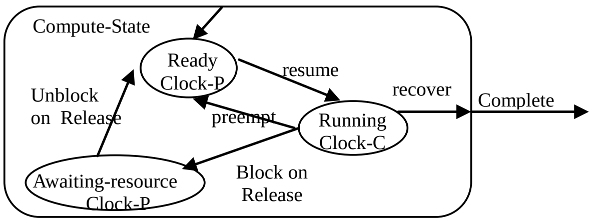


Figure 2. *Compute* state of a Thread.

ship equational theory describing the thread static structure. The Σ_T signature specifies the sorts (and subsorts) set and mainly the operators set necessary to describe each clause of thread description: *features*, *properties*, *modes*, *type* and *implementation*. Operators considered in Σ_T may *freeze* the rewriting under their specified arguments. The *instantaneous* and *tick* rewrite rules R_T (labeled by L_T set elements) describe the behavior of thread dynamic configurations, involving, at each stage, local state of a thread and the state of each one of its connections ports.

As RT-Maude language implements real time rewrite theories, we use it to build the appropriate RT-Maude executable modules. We define a set of generic rules (table 1) mapping each architectural element involved in an AADL architecture description to RT-Maude concepts. As shown in Table 1, an AADL architecture description is mapped to RT-Maude module which may contain many threads declaration (thread class). Thus, formal analysis of behavioral properties with respect of AADL thread declared constraints is defined very naturally.

Table 1. Mapping AADL architectural elements to RT-Maude object-oriented concepts.

AADL Architectural Element	RT-Maude Object-Oriented Concepts
AADL *Architecture*	Real-Time object-oriented rewrite theory
Thread component	*Thread* class
Component *Interface*	*PortState* sort for *IPort* and *OPort* attributes, *File* sort for *AccessData*, *InBufferPort* and *OutBufferPort* attributes
Thread *State*	*ThreadState* sort
Thread *Configuration*	Conditional rewrite rules
Thread *Interaction*	*Message Transmission* between objects (instances of thread class)
Flow Latency in a Thread	Message transmission *time*
Thread *Implementation*	*ThreadImpl* class (subclass of class Thread)
Thread *execution properties*	*Time* sort for *Period* and *Execution-time* attributes

In this more abstract level of formalization, thread type is modeled by the *Thread* class (figure 3), whose attributes are respectively: 1) the functional interfaces *IPort* and *OPort*, 2) data subcomponent, represented as a buffer, related to each connection port *InBufferPort, OutBuffer-Port* and 3) *TState* to specify its state. The *PortState* sort represents the functional interfaces states of a thread. Its internal structure (or its implementation) is specified by the *ThreadImpl* class (figure 3) having the attributes: 1) *Sstate* specify substate of the "*Compute*" composite state, 2) *Time* (predefined sort) specify both temporal execution properties (*Period* and *Execution-Time*) and the *Clock-P* and *Clock-C* clocks for the warning of the thread period and execution time. It is obvious that the *ThreadImpl* class is declared as a subclass of the *Thread* class and profits from the inheritance concept.

```
class Thread | IPort : PortState, OPort : PortState,
TState : ThreadState, InBufferPort : File,  AccessData :
File, OutBufferPort : File,  MaxPreempt : Nat, MaxData :
Nat, NBS : Set .
```
```
class  ThreadImpl  |   Substate  :  Sstate,  Period  :
Time,Execution-time : Time, Clock-P : Time, Clock-C :
Time .
```
```
subclass ThreadImpl < Thread .
subsort Sstate < ThreadState .
```

Figure 3. RT-Maude specification of AADL thread。

We define the passage of data (and/or event) flow through a threads connection as a messages transmission between a thread and its neighbors (threads). Message transmission time is taken into account by the *DlyMsg* sort modeling the flow latency time.

3.2. Defining AADL Architecture in RT Maude

Since in AADL architecture description of a system, the thread is never alone, we have to model thread behaviour inside a larger configuration containing some interconnected threads with the passage of data flow (the message transmission). Besides, the thread temporal execution properties (*Period* and *Compute-Execution-Time*) have also to be considered. Indeed, we first define with constructor operators, the thread states (*wait* or *compute*), the thread substates (*Ready*, *Running*, *Awaiting-Resource*, *Complete* and *noSub*) and the port states (*waitIn*, *waitOut*, r*eceive* and *send*). Then, the behaviour formalization aspect starts with the specification of the visible changes of the thread states, associated to its connection ports and materialized by the rewrite rules *Data-Receive* and *Data-send* (see figure4). The *Data-Receive* rule prepares thread for a new execution period, after receiving a message. It changes the thread state from *wait* to *compute* and its substate from *noSub* to *Ready* initializing the clocks by the values, specified in the execution properties. The *Data-Send* rule putts the thread in its initial state after a period elapse and an execution time. It transforms the thread state from *compute* to *wait* and its substate from *Complete* to *noSub* and then, generates the message with the thread execution result for the transmission.

```
rl[Data-Receive]:(from T1 to T2 transfer DT) < T2:
ThreadImpl | IPort: waitIn, TState : wait, Substate:
noSub , Clock-P: 0, Clock-C: 0, Period: R, Execu-
tion-time: R', InBufferPort: L2 >
=>  < T2 : ThreadImpl | IPort: receive, TState:
compute , Substate: Ready, Period: R,Execution-time:
R', Clock-P: R, Clock-C: R', MaxData: 0, MaxPreempt:
0, InBufferPort:  DT;  L2 >.
```
```
rl[Data-Send]: < T1: ThreadImpl | OPort: waitOut,
TState: compute, Substate: Complete, OutBufferPort:
L; DT, NBS:(T2 & R), Clock-P: R1, Clock-C: R2,
MaxPreempt: N, MaxData: N1 >
=> < T1: ThreadImpl | OPort: waitOut, TState: wait ,
Substate: noSub, OutBufferPort: L , NBS: T2 & R ,
Clock-P: 0, Clock-C: 0, MaxPreempt: 0, MaxData: 0 >
dly(from T1 to T2 transfer DT, R).
```

Figure 4. Specification of AADL connection and interaction.

In this formalization, we take into account the substate of the thread in its active hierarchical state *compute.* We define these substates and their corresponding transitions (see figure 2) and also the corresponding declared properties. The first rewrite rule, in figure 5, changes the thread substate from *Ready* to *Running* and prepares the received data treatment. The rewrite rule *finish* considers the particular case, where the thread doesn't have an out port (*OPort = NoPort*). It gives the thread in its initial state after elapse of the period time. The conditional rewrite rules *resume*, *preempt*, *block-on-Release-Resource*, *Unblock-on-Release-Resource*, *recover* and *complete-rec* define the transitions between substates of the *compute* state. The function *mte* is used to calculate the maximal time elapse of a thread configuration, before a significant action is taken (about the minimum values of the two clocks). Moreover, the function *delta* models the effect of passage of a time *R* on the thread by decreasing one or both of its clocks according to the time elapsed. These two functions are used in the *tick rule* in order to calculate and apply the time elapse on the thread configuration (figure 6). The *nonexec* attribute of the *tick rule* indicates that this rule advances time when no other rule is executable. The *delta* operation modifies only the attributes of sort *time*. The equations calculate the *delta* operation effect on thread configuration and on the messages transmission. The *mte* operation evaluation considers a thread (if it is at compute state) with its clocks initialized by the execution properties values. Then, it considers the distribution of the *mte* operation on the configuration.

```
Crl [init] : < Imp : ThreadImpl |InBufferPort: L,
AccessData: EmptyFile, Substate: Ready, Clock-C: R'>
=>  < Imp: ThreadImpl | InBufferPort: Queu(L),
AccessData: Head(L), Substate: Running,  Clock-C: R' >
if (R' > 0) .
```

```
crl [complete-Rec] : < Imp: ThreadImpl | IPort :
receive , TState :  compute, Substate: subS ,
AccessData: Temp2 , OutBufferPort: L,  OPort: waitOut,
Clock-P: R, Clock-C : R' >
=>  < Imp: ThreadImpl | IPort: waitIn , TState: compute,
AccessData: EmptyFile , OutBufferPort: Temp2 ; L ,
Substate: Complete,  OPort : waitOut , Clock-P : R,
Clock-C : R' >  if (R == 0) .
```

```
rl[finish]: <Imp : ThreadImpl | IPort: receive, TState:
compute, Substate: Complete , OPort: NoPort, Clock-P:
R, Clock-C: R', MaxPreempt: N, MaxData: N1 > =>  < Imp:
ThreadImpl | IPort: waitIn, TState: wait, Substate:
noSub, OPort: NoPort, Clock-P: 0, Clock-C: 0,
MaxPreempt: 0,  MaxData : 0  > .
```

Figure 5. Specification of AADL thread Behaviour.

Tick rule	`crl [tick] : {C:Configuration} => {delta(C:Configuration, R)} in time R if (R <= mte(C:Configuration)) /\(not agerEnabled(C:Configuration)) [nonexec] .`
delta operation	`eq delta(< Imp : ThreadImpl | Substate : Running, Clock-P : R , Clock-C : R' >, R'') =  < Imp : ThreadImpl | Substate : Running, Clock-P : R monus R'' , Clock-C : R' monus R'' > .`
mte operation	`ceq mte(NeC NeC') = min(mte(NeC), mte(NeC')) if (NeC =/= none)  /\ (NeC'  =/= none) .`

Figure 6. The elapsed time on thread configuration.

4. Formal Analysis of Thread Properties

The concurrent execution of several threads and the effect of interactions between them are defined by this ABAReL annex extension of AADL. This section will explain how, under appropriate assumptions, we can model check in RT-Maude behaviour properties AADL architectural system specified as an object-oriented real-time module.

```
ops GPS  TGPS TSCREEN : -> Oid [ctor] .
op initState : -> GlobalSystem .

eq initState = {(from GPS to TGPS transfer data1 ) < TGPS:
ThreadImpl |IPort : waitIn, TState : wait, Substate :
noSub , OPort : waitOut, InBufferPort : EmptyFile,
AccessData : EmptyFile, OutBufferPort : EmptyFile,
Period : 20,  Execution-time : 10, Clock-P : 0, Clock-C :
0, MaxData : 0, MaxPreempt : 0 , NBS : (TSCREEN & 4) >

 < TSCREEN : ThreadImpl | IPort : waitIn, TState : wait,
Substate: noSub, OPort: NoPort, InBufferPort :
EmptyFile, AccessData : EmptyFile, OutBufferPort :
EmptyFile , Period : 15, Execution-time : 7, Clock-P:
0, Clock-C: 0, MaxData : 0, MaxPreempt: 0 , NBS : EmptySet
>}.
```

Figure 7. The GPS example in RT-Maude.

We will deal here with properties such as *reachability*, *safety* and *liveness* through the modeling of a GPS system as case study.

We consider an AADL architectural description modeling a GPS system example. This system should display the current position information for the user. It is composed of one sensor *GPS* and two threads: *TGPS* and *TScreen*. The GPS sensor captures information parameters from satellite and sends them to thread *TGPS*. *TGPS* reads these parameters, converts them into an internal representation, and sends the result to thread *TScreen*. This one displays the recent position received periodically. The AADL specification gives only static description of components and their connections including, for each thread description, the specification of implementation conditions. This is done by the properties declaration, such as: *Dispatch-Protocol*, *Period* and *Compute-Execution-Time* and their various values.

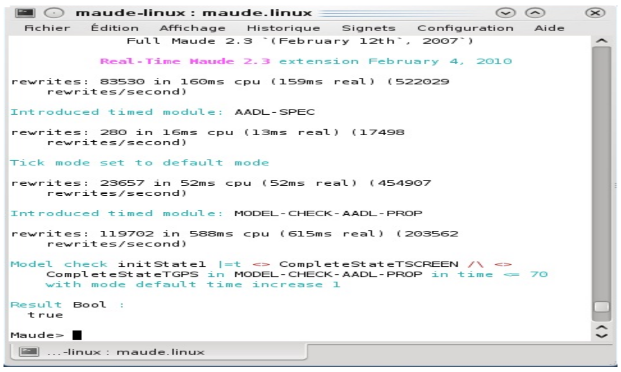
Giving AADL description, we can exploit the extended ABAReL annex to formalize the behavior of each declared thread *TGPS* or *TScreen* and their possible connection. For this case, we associate an RT-Maude module to each thread, specifying its static and dynamic aspects. In order to provide the global system behavior specification, we extend the obtained modules by additional information defining communication between connected threads (*TGPS* and *TScreen*). A code portion of this global RT-Maude module is shown in figure 7. Each thread name is declared as an object identifier. *Initstate* operator gives for each thread its initial configuration thanks to the equation clause. Each thread in *compute* state must have the following substates: *Ready*, *Running* and *Awaiting-Resource*. In this case, we can check for instance the following properties. **(P1)** The final state *(Complete substate)* of the thread execution process can be reached in time. **(P2)**

The thread in execution *(compute state)* is never (i) preempted infinitely, (ii) waiting resource infinitely. **(P3)** The thread will be eventually executed *(running substate)*, respecting the execution time declared in Compute-Execution-Time property. The core module *MODEL-CHECK-AADL-PROP* (figure 8) imports the predefined module *TIMED-MODEL-CHECKER* and the module *AADL-SPEC* to be analyzed. The specification of the previous properties is made through the atomic propositions: *CompleteStateTGPS* **(P1)**, *CompleteStateTSCREEN* **(P2)** and *RunningState* **(P3)**. The check, for instance, of the property **(P1)** by the LTL model-checker of RT-Maude is launched by this command: `(mcinitState1|=t(<>completeStateTGPS)/\(<> CompleteStateTSCREEN) in time <= 70 .)`

```
(tomod MODEL-CHECK-AADL-PROP is
 including TIMED-MODEL-CHECKER .
 protecting AADL-SPEC .
 ops RunningState  CompleteStateTGPS
      CompleteStateTSCREEN : -> Prop [ctor].
 var REST : Configuration .
 var Imp : Oid . vars R R' R'' : Time .
 eq {REST <TGPS:ThreadImpl|Substate: Complete>}
    |= CompleteStateTGPS = true .
 eq {REST  < TSCREEN : ThreadImpl | Substate :
    Complete >} |=  CompleteStateTSCREEN =
    true.
 eq {REST < Imp : ThreadImpl| TState : compute,
Substate : Running   >} |= RunningState = true.
endtom)
```

Figure 8. The MOD*EL-CHECK-AADL-PROP* RT-Maude module.

A direct and naive execution of this command returns a counterexample indicating that the property **(P1)** is not satisfied. To overcome this incorrect behavior, we associate a time interval to the *preempt* transition, so it can be executed no more that a given time. The second successful solution is to add priorities to some rewrite rules execution. The screen shot of figure 9 shows that the property **(P1)** is then evaluated to true in this case. In a similar way, we check the property **(P2)** and **(P3)**.



```
              Full Maude 2.3 `(February 12th`, 2007`)

          Real-Time Maude 2.3 extension February 4, 2010

rewrites: 83530 in 160ms cpu (159ms real) (522029
    rewrites/second)

Introduced timed module: AADL-SPEC

rewrites: 280 in 16ms cpu (13ms real) (17498
    rewrites/second)

Tick mode set to default mode

rewrites: 23657 in 52ms cpu (52ms real) (454907
    rewrites/second)

Introduced timed module: MODEL-CHECK-AADL-PROP

rewrites: 119702 in 588ms cpu (615ms real) (203562
    rewrites/second)

Model check initState1 |=t <> CompleteStateTSCREEN /\ <>
    CompleteStateTGPS in MODEL-CHECK-AADL-PROP in time <= 70
    with mode default time increase 1

Result Bool :
  true

Maude> █
```

Figure 9. Model-check AADL properties.

5. Conclusion

The proposed extension of behavioral annex ABAReL based on object-oriented real-time rewrite theories is a suitable semantic framework for AADL thread behavior description. In this paper, we have exploited this formal setting to define and analyze semantic execution model of AADL architectural system, composed of interconnected threads and juxtaposed messages under timing requirements. Concurrent interactions between the objects are governed by ordinary rewrite rules corresponding to instantaneous transitions and by "tick" rewrite rules describing the time elapse. We showed how this formalism supports the verification of thread temporal execution properties via the LTL model checker tool of RT-Maude system. In future, we will exploit the proposed approach to analyze others properties of the AADL standard, related to space and time partitioning.

REFERENCES

[1] C. Jerad, K. Barkaoui, and A. Grissa-Touzi, 'Hierarchical Verification in Maude of LfP Software Architectures', In ECSA'07, *1st European Conference on Software Architecture*, Aranjuez (Madrid), LNCS , Springer, 2007, pp. 156-170.

[2] SAE International, "Architecture Analysis and Design Language (AADL) Standard", *Version 2, SAE Dtaft Standard AS5506 V2*, 2008.

[3] M. Benammar, F. Belala, , K. Barkaoui, and N. Benlahrache, "Extension d'ABAReL par les Propriétés d'Exécution", *CAL'09, 3ième Conférence Francophone Sur les Architectures Logicielles,* Cépaduès-Editions, RNTI L-4,2009, pp.45-58.

[4] P. C. Ölveczky, "Real-Time Maude 2.3 Manual", Department of Informatics, University of Oslo, 2007.

[5] B. Berthomieu, P.-O. Ribet, and F. Vernadat, "The tool TINA-construction of abstract state spaces for Petri nets and time Petri nets", *International Journal of Production Research*, Vol. 42, 2004, pp. 2741-2756.

[6] Z. Yang, K. Hu, D. Ma, and L. Pi, "Towards a Formal Semantics for The AADL Behavior Annex", *In Design, Automation & Test in Europe DATE 2009*, pp. 1166-1171.

[7] M. Chkouri, A. Robert, M. Bozga, and J. Sifakis, "Translating AADL into BIP -Application to the Verification of Real-time Systems", *In Proc. of MoDELSACES-MB Model Based Architecting and Construction of Embedded Systems*, 2008, pp. 39-54.

[8] O. Sokolsky, I. Lee, and D. Clarke, "Process-Algebraic Interpretation of AADL Models", *14th International Conference on Reliable Software Technologies,* LNCS 5570, 2009, pp. 222-236.

[9] D. Monteverde, A. Olivero, S. Yovine, and V. Braberman, "VTS-based Specification and Verification of Behavioral Properties of AADL Models", *Verimag Research Report n° TR-2008-11*, 2008.

[10] J. Meseguer, "Membership algebra as a logical framework for equational specification", *Recent Trends in Algebraic Development Techniques*, LNCS, Springer Berlin/Heidelberg, Volume 1376, 1998, pp. 18-61.

[11] M. Clavel, F. Duran, S. Eker, N. Marti-Oliet, P. Lincoln, J. Meseguer, and C. Talcott, *Maude Manual (Version 2.4)*, 2008.

[12] P. Dissaux, J.P. Bodeveix, M. Filali, P. Gaufillet, and F. Vernadat, "AADL Behavioral Annex", *Pro. of the DASIA'06, Data Systems In Aerospace- Conference, Berlin, Germany, ESA SP-63*, 2006.

[13] P. Gaufillet, "TOPCASED-Toolkit In Open source for Critical Applications & systems Development", *AADL Workshop*, 2005.