Towards a Formal Semantics for the AADL Behavior Annex

Zhibin Yang, Kai Hu, Dianfu Ma
School of Computer Science and Engineering
Beihang University
Beijing, P.R.China
kenney@cse.buaa.edu.cn, {hukai,dfma}@buaa.edu.cn

Lei Pi
IRIT
CNRS, Université Paul Sabatier
Toulouse, France
pilei@irit.fr

***Abstract*—AADL is an Architecture Description Language which describes embedded real-time systems. Behavior annex is an extension of the dispatch mechanism of AADL execution model. This paper proposes a formal semantics for the AADL behavior annex using Timed Abstract State Machine (TASM). Firstly, the semantics of AADL default execution model is given, and then we formally define some aspects semantics of behavior annex. A prototype of real-time behavior modeling and verification is proposed, and finally, a case study will be given to validate the feasibility.**

Keywords- AADL; behavior annex; execution model; TASM

I. INTRODUCTION

AADL (Architectural Analysis and Design Language) [1] is an Architecture Description Language standardized by the SAE (Society of Automotive Engineers) in November 2004. It is especially effective for model-based analysis and specification of complex embedded real-time systems (ERTs).

Specification of system real-time behavior is one of major concern for time-critical systems. Although some behavioral aspects can be described with the core of the AADL standard, such as mode change, actual behaviors of components rely on target source code. The AADL behavior annex [2] proposed by IRIT in 2006, is an extension of AADL to offer a way to specify the behaviors of components without expressing them with the target language, therefore it can support more precise behavioral and timing analysis. However, it must rely on a well defined formal semantics. This paper proposes to describe its formal semantics using TASM (Timed Abstract State Machine) [3] language.

Furthermore, AADL execution model precisely defines runtime real-time patterns such as dispatch, communication and timing of components. Behavior annex is an extension of the dispatch mechanism of execution model, to describe the actual behaviors of components in detail, and execution model specifies when the behavior annex is executed and on which data it is executed. We can say execution model is the basic or context of behavior annex. So the semantics of execution model also need to be formalized.

TASM extends the Abstract State Machine formalism to enables the explicit expression of timing, resource, communication, composition, parallelism, etc. Timing semantics of execution model and behavior annex is the main topic in this paper, and will give properties verification based on the TASM model.

II. NEED FOR PRECISE SEMANTICS OF AADL BEHAVIOR ANNEX

A. AADL overview

AADL employs formal modeling concepts for the description of software/hardware architecture and non-functional properties of ERTs in terms of distinct components and their interactions. AADL offers a set of predefined component categories [1].

- Thread, thread group, subprogram, data and process.
- Processor, memory, bus and device.
- System represents composite sets of software and execution platform components.

For instance, a thread represents a sequential flow of execution and it is the only AADL component that can be scheduled. A subprogram represents a piece of code that can be called by a thread or another subprogram.

Communication between threads can be realized through dataflow, call to server subprogram or access to shared variable. These various connection points are declared in the interface of the communicating components and are called features. They will be Ports, Server Subprograms or Data Access depending on the chosen communication paradigm.

B. AADL execution model

System behaviors do not only rely on the architecture defined by such above components and their connections but also rely on the runtime environment [4]. AADL standard has specified execution model as a virtual runtime environment, which contains synchronous as well as asynchronous patterns, to support the execution and management of components. The synchronous pattern consists of periodic threads with data ports communication. To this model are added asynchronous features: events which may be associated with data can be sent through the *Raise_Event* system call at any time and may result in the dispatch of destination threads. Timing aspects such as deadline, dispatch time, are also defined in the execution model, declared through AADL properties.

The described work is funded by the National High-Tech Research and Development Plan of China under Grant No. 2006AA01Z19A, and Aviation Science Foundation of China.

978-3-9810801-5-5/DATE09 © 2009 EDAA

We only consider synchronous execution model in this paper, so we express periodic thread and data port communication in detail in the following.

1) Periodic thread

AADL defines a thread can be *stopped*, *active*, and *inactive*, an active thread can be waiting for *dispatch*, *read data*, *computing*, and *write data*. Several properties can be assigned to threads, like listing 1, the *Dispatch_Protocol* is supposed to be *Periodic*, with period given by the *Period* property in the form of period=20ms or frequency=50Hz, its *Deadline*, and *Execution Time*.

Listing 1 Periodic thread

```
thread implementation T1
properties
Dispatch_Protocol => Periodic;
Period => 20ms ;
Deadline => 20ms ;
Compute_Execution_Time => 9ms ;
end T1;
```

2) Data port communication

In order to ensure deterministic data communication, AADL offers two communication mechanisms: immediate and delayed, as shown in Fig.1. For an immediate connection the execution of the recipient thread is suspended until the sending thread completes its execution and makes its output available to the recipient. For a delayed connection the output of the sending thread is not transferred until the sending thread's deadline, typically the end of the period. In other words, its output is not available to the recipient until the next dispatch. Furthermore, AADL considers three cases between two periodic threads: synchronous (with the same period), oversampling (the period of receiver is evenly divided by the period of sender), and undersampling (the period of sender is evenly divided by the period of receiver).

So, there are six compositional patterns for data port communication in AADL. They can not only ensure deterministic and consistent data communication, but also support schedulability analysis and timing verification.

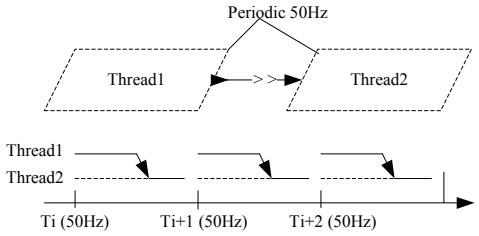

Figure 1(a) Immediate connection

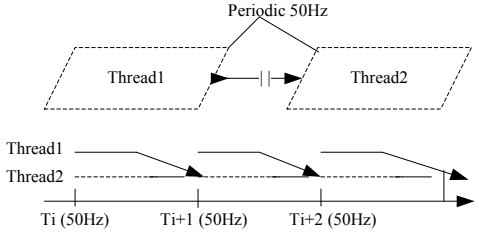

Figure 1(b) Delayed connection

C. Behavior annex

Behavior annex lies in the *computing* state, is an extension of the dispatch mechanism of execution model, to describe more precisely the behaviors such as port communication, subprogram call, timing, asynchronous, etc. and the AADL execution model specifies when the behavior annex is executed and on which data it is executed. A full AADL model should contain well-defined structure, execution model and behavior annex. Now, a behavior annex can be attached to a thread and a subprogram. It is described using an extension of AADL mode automata [4]: initial to specify a start state, return to specify the end of a subprogram or complete to specify completion of a thread, transitions may be guarded by conditions and actions, conditions and actions include sending or receiving events, calling or executing subprograms, assigning or testing data variables as well as execution abstractions such as use of CPU time or delay. There are many presentations of the annex already exist, but it is difficult to represent its formal definition with an adequate level of abstraction. So we just introduce it at a syntactic level through an example:

Listing 2 Behavior annex example

```
subprogram implementation example.i
annex behavior_specification {**
states
s0: initial state;
s1: return state;
transitions
s0 -[p?(x) ]-> s1 { p!(x+1); };
**};
end example.i;
```

D. Semantics problems and analysis

For execution model, we just consider synchronous one, i.e. periodic threads with immediate and delayed communication. For behavior annex, we will give subset of it semantics contains send/receive event, remote subprogram call, asynchronous, timing, etc. Fig.2 shows the relation between execution model and behavior annex.

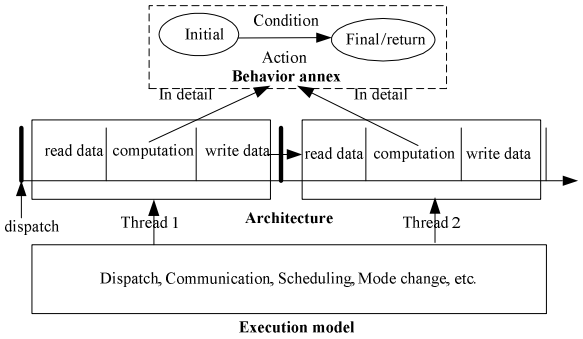

Figure 2 Behavior annex and execution model

III. A FORMAL SEMANTICS FOR AADL BEHAVIOR ANNEX

A. A brief presentation of TASM

TASM extends the ASM formalism to enables explicit expression of timing, resource, communication, composition, parallelism etc. A basic TASM specification contains an abstract state machine and an environment. The environment contains environment variables and the universe of types that variables can have. The machine consists of three parts –

monitored variables, controlled variables, and mutually exclusive rules with the form of "*if condition then action*". An *action* is a sequence of one or more updates to environment variables. So, system behavior can be specified as the rules of the machine. We will present its main characteristics, more detailed information about this language may be found at [5].

- Provides the concepts of main machine, sub machine and function machine to support the specification of *hierarchical* behaviors.
- Uses a set of main machines that execute in parallel, to support the specification of *parallel* behavior.
- *Time* specification is to specify the duration of a rule execution, which can take the form of a single value, an interval [*tmin*, *tmax*], or a keyword *next*. The interval is useful to capture the uncertainty and the semantics of rule execution are that of a delay in the current state followed by an instantaneous state update. It uses relative time between rules, that is, the total time is simply the summation of the individual rule times. It also supports hierarchical time semantics, the time will be composed to reflect the duration of the longest update set, and parallel time semantics, regards the synchronization of the main machines with respect to the global progression of time.
- A *resource* is defined as a global quantity that has a finite size such as processor usage, memory, and bandwidth, etc. Each rule specifies how much of a given resource it consumes, either as an interval or as a single value, the semantics of resource usage are assumed to be volatile, that is, the usage lasts only through the duration.
- *Communications* are defined as in the Calculus of Communication Systems (CCS) model,enables explicit specification of dependencies between machines.

In brief, the execution model of TASM is a loop: read input variables, wait for the duration of the execution, write output variables, and wait for synchronization. It is very important for the semantics specifications.

B. Modeling Periodic threads

1) Real time concepts of periodic threads

A periodic thread can trigger a next dispatch at the instant of period, and complete in the deadline. According to TASM execution model and parallel time semantics, we introduce a main machine to manage period and another one represents the execution of thread with deadline, WCET, resource and execution rules. WCET represents execution time and resource represents the CPU utilization following the formula: processor = WCET*100/deadline. We suppose here that a thread uses CPU uniformly between its dispatch and deadline, which corresponds to using a preemptive scheduler with arbitrary high task switch speed. The system is schedulable if processor usage is always bounded by 100. In TASM, a state is defined as the values of all variables at a specific step, so we define environment variables with values as follow: *Condition:={true, false}*, *Nextdispatch:={true, false}*, *Complete:={true, false}*.

Listing 3 TASM specification of periodic thread

```
MAIN MACHINE: Period
Rule: period
{ t := period ;
  if Nextdispatch =false then
  Nextdispatch:=true;
}
```

```
MAIN MACHINE: Execution
Rule: execution
{ t := [0,deadline] ;
  processor := WCET*100/ deadline ;
  if Complete =false then
  Complete:=true;
}
```

2) Communication semantics of periodic threads

TASM explicit time model with powerful *logical* time constraints allows specifying precisely the scheduling aspects of threads, parallel and communication mechanisms can be used to describe the synchronization.

From the Fig.2, the two periodic threads (named *sender, receiver)* are associated with clocks (*tsender* and *treceiver* respectively). These clocks, purely logical, represent the dispatches of the threads. Using *tsender, tsenderexestart, tsenderexecomplete, tsenderwrite, tsenderdeadline* and *treceiver*, *treceiverexestart*, *treceiverexecomplete*, *treceiverread*, *treceiverdeadline* to describe dispatch time, start of execution, end of execution, transformation data and deadline of *sender* and *receiver* respectively. Let *ssender* and *sreceiver* be natural numbers such that *fsender*/*freceiver*= *ssender*/*sreceiver*, *fsender*, *freceiver* represent the frequency of the two threads. When the threads are synchronous *ssender* = *sreceiver*= 1, then *tsender*= *treceiver*, this means *sender* and *receiver* dispatch simultaneously. When oversampling, *ssender*= 1 and *sreceiver*> 1, then *treceiver*/*tsender* = *sreceiver*, this means each instant of *tsender* is synchronous with every (*sreceiver*)th instant of *treceiver*. When undersampling, *ssender*> 1 and *sreceiver*= 1, then *tsender*/*treceiver* = *ssender*, means each instant of *treceiver* is synchronous with every (*ssender*)th instant of *tsender*.

Immediate communication, the dispatch time for the two threads is the same, so the receiver has read old data before communication, a *Buffer* machine is introduced to *Replace* the old data. In the case of synchronous, the threads have the same period and complete in a period, *tsenderwrite* is synchronous with *tsenderexecomplete*, Channel *Chal* is defined to express communication. *Chal?*, *Chal!* mean receive and send message. TASM defines the operator ⊙ for parallel composition, so the semantics can be specified as Period ⊙ Sender ⊙ Buffer ⊙ Receiver.

Listing 4 Immediate communication with synchronous

```
MAIN MACHINE: Period
MAIN MACHINE: Sender
Rule1: sender_execution
{ t := tsenderexecomplete;
  processor := WCET*100/ deadline ;
  if Complete =false then
  Complete:=true;
}
Rule2: write_data
{  t:=0;
  if Condition=true then
  Chal!;
}
```

```
MAIN MACHINE Buffer
Rule: displace_old_data
{  t:= treceiverread;
  if Condition=true then
   Replace:=true;
}
MAIN MACHINE: Receiver
Rule 1: read_new_data
{  t:= treceiverexestart;
  if Condition=true then
    Chal?;
}
Rule 2: receiver_execution
{ t := (treceiverexecomplete
   - treceiverexestart);
  if Complete =false then
   Complete:=true;
}
```

Delayed communication, the value from the sending thread is transmitted at its deadline and is available to the receiving thread at its next dispatch. So the receiver just needs the old data from sender. We assume the deadline of the sending thread and the dispatch of the receiving thread occur simultaneously, so the transmission occurs at that instant.

In the case of synchronous, *tsenderwrite* is synchronous with the next dispatch of sender (i.e. *tsenderdeadline*). We just give some main TASM expressions for compactly in the following sections.

Listing 5 Delayed communication with synchronous

```
MAIN MACHINE: Sender
Rule 1: sender_execution
Rule 2: write_data
{  t:= (tsenderdeadline-
   tsenderexecomplete);
   if Condition=true then
   Chal !;
}
```

```
MAIN MACHINE: Receiver
Rule 1: read_data
{  t:= (treceiverdeadline+
   treceiverexestart);
   if Condition=true then
   Chal ?;
}
Rule 2 : receiver_execution
{ t := (treceiverexecomplete
   - treceiverexestart);
  if Complete =false then
   Complete:=true;
}
```

For oversampling, *tsenderwrite* is also synchronous with the next dispatch of sender, but there will be *sreceiver* executions of receiver occurring in one execution of sender. So the time of read data is shown as the following equation.

$$t_i = tsenderdeadline + (i-1)treceiverdeadline + treceiverexestart \quad (i=1..\ sreceiver) \qquad (1)$$

Listing 6 Delayed communication with oversampling

```
MAIN MACHINE: Sender
Rule 1: sender_execution
Rule 2: write_data
{  t:= (tsenderdeadline-
   tsenderexecomplete);
   if Condition=true then
   Chal !;
}
```

```
MAIN MACHINE: Receiver
Rule 1: read_data(i)
  { t:=ti;
   if Condition=true then
   Chal ?;
}
Rule 2 : receiver_execution(i)
{ t := (treceiverexecomplete
   - treceiverexestart);
  if Complete =false then
   Complete:=true;
}
```

For undersampling, there will be *ssender* executions of sender occurring in one execution of receiver, but only the most recently transmitted data is available. The complete time of sender is shown as the following equation.

$$t_j = (j-1)tsenderdeadline + tsenderexecomplete, \quad (j=1..ssender) \qquad (2)$$

Listing 7 Delayed communication with undersampling

```
MAIN MACHINE: Sender
Rule 1: sender_execution(j)
{ t :=tj;
  processor := WCET*100/ deadline ;
  if Complete =false then
  Complete:=true;
}
Rule 2: write_data(j)
{  t:= (tsenderdeadline-
   tsenderexecomplete);
   if Condition=true then
   Chal !; }
```

```
MAIN MACHINE: Receiver
Rule 1:  read_data
{t:=((ssender)tsenderdeadline
   +treceiverexestart);
   if Condition=true then
   Chal ?;
}
Rule 2 : receiver_execution
{ t := (treceiverexecomplete
   - treceiverexestart);
  if Complete =false then
   Complete:=true; }
```

C. *Formal semantics of behavior annex*

Behavior annex describes system behaviors in the following form:

<state> -[<guard>?]-> <state> [{<action>*}]

As the hierarchical composite mechanism in TASM, *State* can mapping to controlled/monitored variables, *guard* can be specified using function machines with Boolean output parameter, *action* can also be specified using function or sub machines, and the whole transition can be expressed by rules of a main machine.

In detail, guards and actions can include sending/receiving events, calling/executing subprograms, execution abstractions such as use of CPU time or delay, as well as composite and concurrency state.

1) Send/Receive events

Default execution model just consider periodic threads with data port communication. But behavior annex can access event or event data port variables, produce data and send it on out ports, event and event data ports are associated to queues. Parallel composite in TASM enables the specification of both synchronous and asynchronous system, so we can use two main machines in parallel, and define another main machine for dealing with the queues. This is similar with the former communication semantics.

2) Remote subprogram Call

Remote subprogram calls concern two threads and one subprogram. A client sends the request and waits for the result. The server receives requests and remote call the subprogram; this is a three-way synchronous.

Behavior annex specifying the subprogram implementation has one or more return states indicating the return to the caller. Values are transmitted before the call for the parameters in IN mode and after the transition to a return state for the OUT mode.

Client-server synchronization can specified by two main machines, we can use a function machine to specify subprogram behavior and define controlled/monitored variables: *ServerASM:={listening, ack}, ClientASM:={request, waiting, resume, execution }, InputParam:={true, false}, OutputParam:={true, false}.*

Listing 8 Remote subprogram call

```
FUNCTION MACHINE Subprogram
Rule: subprogram_execution
{ if InputParam=true then
  OutputParam:=true;
}
MAIN MACHINE Client
Rule: client_request
{if ClientASM=request then
  Chal!;
  ClientASM:= waiting;
}
```

```
MAIN MACHINE Server
Rule 1: server_call
{if ServerASM =listening then
   ServerASM:=ack;
   Chal?;
   InputParam:=true;
   Subprogram();
 }
 Rule 2: client_resume
 { if OutputParam=true then
   ClientASM:=resume;
 }
```

3) Client-server protocol

In AADL execution model, remote subprogram calls are used for synchronous communications. Moreover, in behavior

annex, three communication protocols are extended for remote subprogram call: asynchronous (ASER), synchronous (HSER) and semi-synchronous (LSER).

With the HSER protocol, the caller waits for the completion of the request and gets the results produced by the subprogram called by the server. With the LSER protocol, the caller waits for the acceptation of the request. Then the server calls the corresponding subprogram. With the ASER protocol, the client sends the request and continues its execution.

A function machine *Buffer_RE* is introduced to manage the requests, and just give the TASM specifications of LSER and ASER.

Listing 9 Client-server protocol

```
MAIN MACHINE LSER                MAIN MACHINE ASER
Rule1: request                   Rule 1: ClientASM(i)
{                                {if ClientASM(i)=request then
if ClientASM=request then          Chal!;
Chal!;                             Buffer_RE(i);
ClientASM:=waiting;                ClientASM(i):= execution;
}                                }
Rule2: semi-synchronous          Rule 2: ClientASM(j)
{                                {if ClientASM(j)=request then
if ServerASM.=ack then             Chal!;
ClientASM:=resume;                 Buffer_RE(j);
}                                  ClientASM(j):= execution;
                                 }
```

4) Time behavior

For more precise definition of communication timings, behavior annex denotes *computation(min,max)*, and *delay(min,max)*, which expresses the use of the CPU and suspension for a possibly non-deterministic period of time between min and max respectively. The timing action is related to the transition and not to the states, so this is consistent with TASM semantics. We define a controlled/monitored variable: *CurrentState:={Current, Next}*.

Listing 10 Time behaviors

```
MAIN MACHINE Computation         MAIN MACHINE Delay
Rule : computation(min, max)     Rule: delay(min, max)
{ t:= [min, max]                 { t:= [min, max]
  processor := WCET*100/ deadline ;  if CurrentState= Current then
  if CurrentState=Current then     CurrentState:= Next;
  CurrentState:=Next;            }
}
```

IV. VERIFICATION AND ANALYSIS

A. Framework

We propose a framework (Fig. 3) for modeling, semantics specification and verification of the AADL specification. Osate[6] is an open source AADL tool environment, and support behavior annex modeling with Osate-BA plug-in. Model transformation languages ATL (Atlas Transformation Language) [7] is used as the automatic transformation engine to express the mapping from AADL model to TASM. Then run the model checker UPPAAL to verify timing properties and TASM Toolset [8] to analysis resource consumptions.

This paper just considers the properties verification preliminarily. The model transformation is based on Meta model level and a mapping method from the TASM to UPPAAL has been undertaken [9].

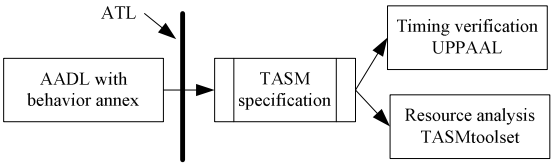

Figure 3 Verification and analysis framework

B. Case study

The simple example illustrates the use of AADL with behavior annex and TASM to model and analysis cruise software for an automotive (Fig. 4). The system is split into system component (*s_cruise*), process component (*cruise*), and several thread components (*command, speed, wheel, throttle, display*). *Wheel* is driven by external event; user use *command* to set a required speed and the required speed is sent to *throttle*. *Speed* component read the tours of the *wheel* and compute the actual speed, then send it to the *throttle*. According to the required speed and the actual speed, *throttle* compute the voltage used to control the automotive.

Wheel is a sporadic thread; other threads are periodic with data port connection, which contains immediate and delayed case. Each thread has a behavior annex specification. We just show *throttle* thread as an example.

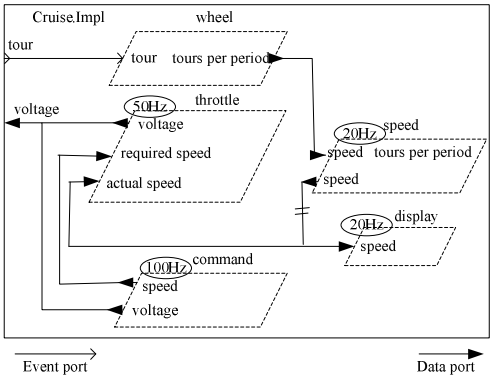

Figure 4 AADL graphic model of cruise system

Listing 11 AADL model of throttle thread and connection

```
thread throttle
properties
Dispatch_Protocol => Periodic;
Period => 20 ms;
Deadline => 20ms ;
Compute_Execution_Time => 9ms ;
end throttle;
thread implementation throttle.i
annex behavior_specification {**
states
s0: initial complete state;
transitions
s0 -[ ]-> s0 { voltage := required_speed - actual_speed; };
**};
end throttle.i;
process implementation cruise.i
connections
-- Immediate connection
data port command.speed -> throttle.required_speed;
--Delayed connection
data port speed.speed ->> throttle.actual_speed;
data port speed.speed ->> display.speed;
end cruise.i;
```

The behavior annex of *throttle* thread in TASM is shown in the listing 12.

Listing 12 TASM specification of throttle thread

```
MAIN MACHINE Throttle
Rule throttle_behavior
{ t:=9;
 processor:=45;
 if s = s0 then
 Chanl?;
 Reqspeed:= required_speed;
 Actspeed:= actual_speed;
 voltage := Reqspeed - Actspeed;
}
```

The verification is achieved with UPPAAL by mapping each main machine to a timed automaton, shown in Fig 7. In UPPAAL, time elapses in a state, but time is used to denote the duration of a transition in TASM. This can be expressed using timed automata with an extra intermediate location to elapse time, like *throttle* location. The *pivot* is the initial location and depicts that the corresponding machine is idle, that is, waiting to execute a rule. Timing correctness can be defined as a reachable state of the system being reachable within an acceptably bounded amount of time. These properties can be express in temporal logic formula CTL.

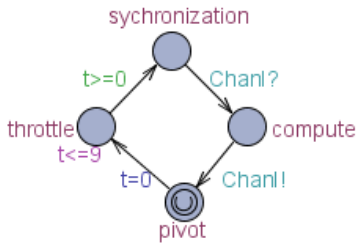

Figure 5 Mapping TASM to UPPAAL's timed automata

V. Related Work

Reference [10] proposes an expression of AADL data port communication using UML MARTE with explicit time model, but do not refers behavior annex. [11] proposes a translation from AADL to BIP taking into account threads, processes and processors as well as the behavioral annex, but do not present the AADL communication protocols. [12] uses TLA+ to specify subset semantics of the AADL execution model, such as preemptive scheduling, communication through ports and shared data. [13] gives the expression of behavior annex using Maude, but not fully, the detailed verification analysis is also not given. FIACRE [14] is a intermediate language for verification of high level models, but it has less powerful constructs of resource.

VI. Conclusions and Future Work

TASM integrates behavior, time and resource into a single specification, that's why we have presented the AADL semantics using TASM. The relation between behavior annex and execution model was presented clearly, in which execution model was considered as the context of behavior annex. So, the semantics of execution model was also given.

Real-time concepts and communication semantics of execution model were presented firstly. Then, subset semantics of behavior annex was described. Furthermore, a verification example was given. The semantics model can be used to be the understanding of the complex mechanisms of AADL behavior annex and execution model and also considered as the basic for schedulability analysis and code generation.

Currently, we are concerning more complicated semantics, such as aperiodic thread with event port, end-to-end flow and mode change in execution model, and other mechanisms in behavior annex. The extension of TASM, automatic model transformation tool and more precise analysis methods will be our future work.

Acknowledgements

We would like to thank Prof. Mamoun Filali, Prof. Jean-Paul Bodeveix from IRIT in France and Prof. Zonghua GU from Hong Kong University of Science and Technology in P.R.China.

References

[1] SAE Aerospace. SAE AS5506: Architecture Analysis and Design Language (AADL), Version 1.0, 2004.

[2] SAE AS5506 Annex: Behavior_Specification V2.0 September 20, 2007.

[3] Ouimet, M. and Lundqvist, K. The Timed Abstract State Machine Language: An Executable Specification Language for Reactive Real-Time Systems. In Proceedings of the 15th International Conference on Real-Time and Network Systems (RTNS '07), Nancy, France, 2007.

[4] P.Dissaux, J.P. Bodeveix, M. Filali, P. Gaufillet, F. Vernadat. AADL Behavioral annex. Proceedings of the DASIA 2006 – DAta Systems In Aerospace.

[5] M. Ouimet and K.Lundqvist. The TASM Language Reference ManualVersion 1.1. November , 2006.

[6] The SEI AADL Team: An Extensible Open Source AADL Tool Environment (OSATE), Software Engineering Institute, CMU. 2006.

[7] Jouault.F, and Kurtev.I . Transforming Models with ATL. MoDELS 2005, Lecture Notes in Computer Science . pp.128-138.

[8] Ouimet. M. and Lundqvist. K. The Timed Abstract State Machine Toolset: Specification, Simulation, and Verification of Real-Time Systems. In Proceedings of the 19th International Conference on Computer-Aided Verification (CAV '07), 2007.pp 126-130.

[9] M. Ouimet and K. Lundqvist. Verifying Execution Time using the TASM Toolset and UPPAAL, MIT Technical Report 2007.

[10] Charles Andr´e, Fr´ed´eric Mallet, Robert de Simone. Modeling of immediate vs. delayed data communications: from AADL to UML MARTE. ECSI Forum on specification & Design Languages (FDL), September 2007..

[11] M.Y.Chkouri, A.Robert, M.Bozga, and J.Sifakis. Translating AADL into BIP - Application to the Verification of Real-time Systems. The proceedings of the ACESMB 2008 workshop conjunction with MODELS 2008.

[12] J.P.Bodeveix, M.Filali and J.F.Rolland. AADL execution model semantics, AADL communication semantics in TLA, Technical Report, IRIT, 2007.

[13] M. Belala, F.Latreche, F.LATRECHE. AADL behavioral annex based on generalized rewriting logic. Research Challenges in Information Science, 2008. pp.1-8.

[14] B. Berthomieu, J. P. Bodeveix, P. Farail, M. Filali, H. Garavel,P. Gaufillet, F. Lang, and F. Vernadat. Fiacre: an intermediate language for model verification in the topcased environment. Proceedings of the 4th European Congress on Embedded Real-Time Software ERTS'08, January 2008.