Executable AADL

Real Time Simulation of AADL Models

Pierre Dissaux[1], Olivier Marc[2]

[1]Ellidiss Technologies, Brest, France.
[2]Virtualys, Brest, France.
pierre.dissaux@ellidiss.com
olivier.marc@virtualys.com

Abstract. The Architecture Analysis and Design Language (AADL) standard [2] defines a default runtime semantic for software intensive Real Time systems. This includes support for multi tasking, network distributed architectures and Time and Space Partitionning systems. A proper implementation of the AADL runtime thus allows for the virtual execution of a system at a model level and contributes to the early verification of critical software applications.
This paper describes an implementation of the AADL runtime by the Marzhin Multi Agent simulator that is embedded in the AADL Inspector tool [5].

Keywords. AADL, Simulation, Multi Agent

Introduction

The Architecture Analysis and Design Language (AADL) standard defines a default runtime semantic for software intensive Real Time systems. This includes support for multi tasking, network distributed architectures and Time and Space Partitionning systems (TSP). A proper implementation the AADL runtime thus allows for the virtual execution of a system at a model level and contributes to the early verification of critical software applications in the development life-cycle.

This paper firstly summarizes the definition of the default AADL runtime, then describes one of its implementations that has been performed to develop the Marzhin Multi Agent simulator, and finally explains how it can be used in practice within the AADL Inspector tool.

1 The AADL Runtime

The AADL Language is an international standard of the SAE (AS-5506B) [1]. The standard defines in particular a default execution model that specifies the way the various components interact at run-time. This enables a precise timing analysis and simulation of AADL models.

A typical AADL model is composed of one or several execution resources (Processors) that can communicate via Buses. The software application is composed of one or several memory address spaces (Processes) that contain concurrent Threads and shared Data. Various inter-threads communication paradigms are supported.

1.1 Processors

In AADL, the Processor represents the association of a hardware computation resource and a scheduler. It must declare a Scheduling_Protocol property whose value corresponds to one of those that are actually supported by the analysis, simulation or code generator. Typically supported Scheduling_Protocols are:

- Rate Monotonic protocol (RM), based on the period of the Threads.
- Deadline Monotonic protocol (DM), based on the deadline of the Threads.
- POSIX 1003 (HPF), based on the predefined priority of the Threads.
- ARINC 653, for the static scheduling of partition slots.

In the case of a partitioned system, the Processor computation resource is shared between several Virtual Processors, each of them being associated with a set of Threads located in an AADL Process. Virtual Processors must also define their own Scheduling_Protocol property. This is typically what happens when the ARINC 653 Annex of the AADL standard is used.

1.2 Threads

The default behavior of AADL Threads is specified in the standard by a state-transition automaton.

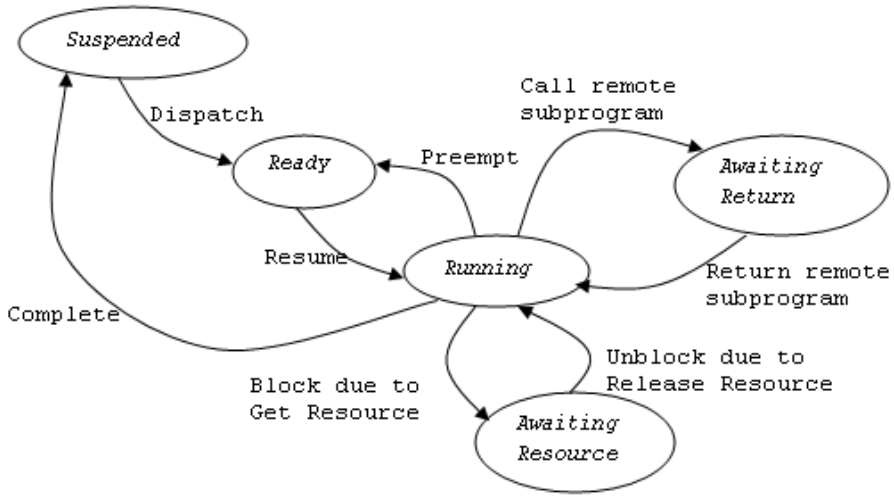

Fig. 1. Runtime states for AADL threads

A Thread must have a Dispatch_Protocol property that defines when it is ready to execute. Supported protocols are:

- Periodic: the thread is periodically triggered by a system clock.
- Aperiodic: the thread is triggered upon arrival of an event on one of its ports.
- Sporadic: same as Aperiodic with a minimum inter-arrival time.
- Timed: same as Aperiodic with an additional timeout event.
- Hybrid: the thread is triggered by event ports and the system clock.
- Background: the thread is triggered when the execution resource is free.

Thread interfaces contain Features that are used to implement communication channels. They can be:

- Data Ports: allows for point to point data exchange
- Event Ports or Event Data Ports: allows for events and message exchange
- Access to shared Data: allows for multi-points data exchange with concurrency control.
- Access to remote Subprograms: allows for remote subprogram calls.

1.3 Shared Data

One particular way to exchange information between Threads is to let them have access to the same shared data. Shared data are represented in AADL by Data subcomponents to which Threads can have access through Data Access Connections.

It is possible to specify critical sections thanks to the AADL Behavior Annex. In order to ensure mutual exclusion of all the threads accessing a given shared data component, a Concurrency_Control_Protocol property can be set. A typical value for this property is Priority_Ceiling_Protocol (PCP).

1.4 AADL Behavior Annex

The core definition of AADL deals with the architectural description of the system. It specifies which components are instantiated and how they are connected and bound together. The functional activity of Threads or Subprograms is summarized by a Compute_Execution_Time property that must be given with its Min and Max values. The Max value of this property thus corresponds to the usual WCET (Worst Case Execution Time) that is used for scheduling analysis.

However, in order to perform precise timing analysis or simulations, it is necessary to provide a more detailed description of the functional behavior of Threads and Subprograms. The AADL Behavior Annex is an action language that can be used to provide a simplified representation of the sequential source code structure (pseudo-code).

Examples of actions that can be defined with the AADL Behavior Annex are:

- `p!` : sending an event on port p (Put_Value and Send_Output)
- `d!<` : entering a critical section on shared data access d (Get_Resource)
- `d!>` : leaving a critical section on shared data access d (Release_Resource)

- `computation(a..b)` : use of the processor for a duration between the minimum duration a and the maximum duration b.

2 The Marzhin Simulator

2.1 Principles of the Marzhin Simulation.

Marzhin is a simulation engine that is based on a multi-agent kernel which implies a random order of activation of the execution units. Each agent can contain one or more execution units that are invoked randomly during a simulation cycle. The agents can be specified independently which highly facilitates the initial development and the maintenance of the simulator. At the execution time, all the agents interact together to exhibit a global behavior.

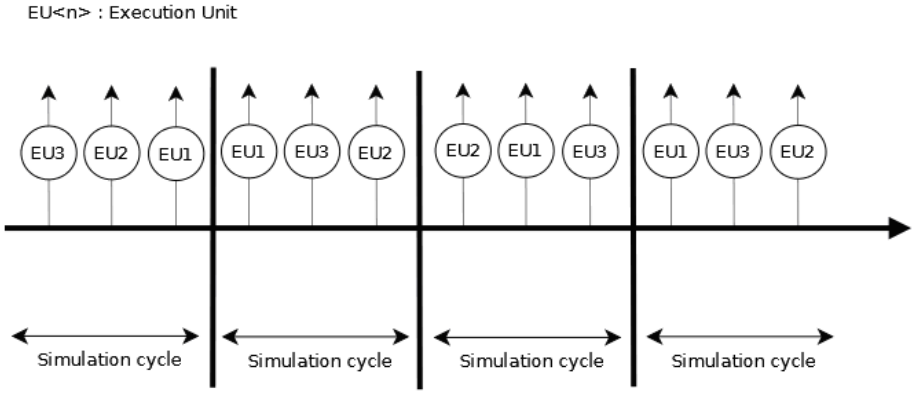

Fig. 2. Marzhin random execution principle

Each AADL entity that is managed by Marzhin is modeled by a specialized agent containing the appropriate execution units to reflect the AADL runtime semantics.

2.2 The Marzhin Data Model.

The Marzhin agents have been defined so that they usually match the corresponding AADL entities. However, in order to well distinguish them from their AADL equivalent, the name of the Marzhin entities has been capitalized in the next paragraphs:

- THREAD: It has the Dispatch_Protocol, DispatchOffset, DispatchJitter, Period, Priority, Deadline and Quantum properties. ThePriority is generally determined by the Scheduling_Protocol property of the PROCESS containing the THREAD. It maintains a list of instructions to be executed. They can be for example: event sending or conditional branches. Their different execution states are the same as those of the AADL Threads as described in section 1.2.

- PROCESS: It represents the address space partitionning and the scheduler. In particular, it thus implements the Scheduling_Protocol that is specified by the AADL Processor.
- PROCESSOR : it contains PROCESSes and manages their scheduling in case of a multi-partition system.

2.3 Marzhin Simulation Cycle.

In the case of the execution of THREADs in a PROCESS, the Marzhin simulation cycles run as follows:

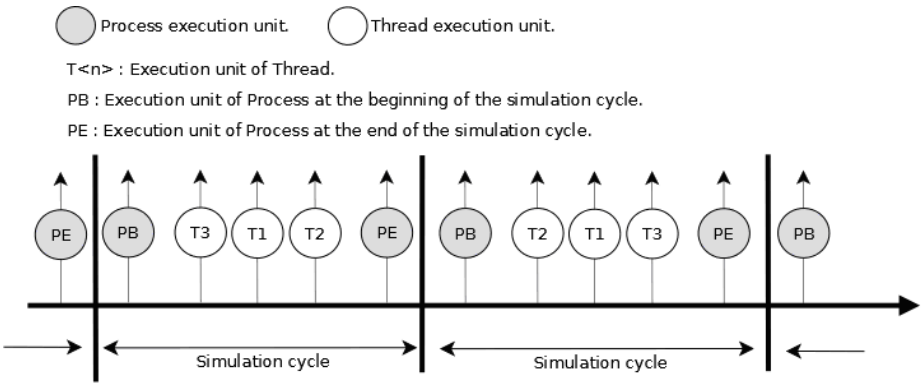

Fig. 3. Marzhin simulation cycle

1. An execution unit starts the simulation cycle of the PROCESS (PB). This allows for updating the priority of each THREAD at each simulation cycle if needed.
2. Execution in a random order of the election process for all the THREADS (T1, T2, T3) in order to update their current internal state and determine the highest priority THREAD that will be executed.
3. An execution unit ends the simulation cycle of the PROCESS (PE) and actually executes the current instruction of the selected THREAD.

In the case of a PROCESSOR containing several PROCESSes, the execution is defined according to the partition slots. If a partition is not active, all the execution units of involved entities (PROCESSs, THREADs ...) are disabled and are not taken into account in the simulation cycle. Only the execution units of the active partition will be activated during the cycle.

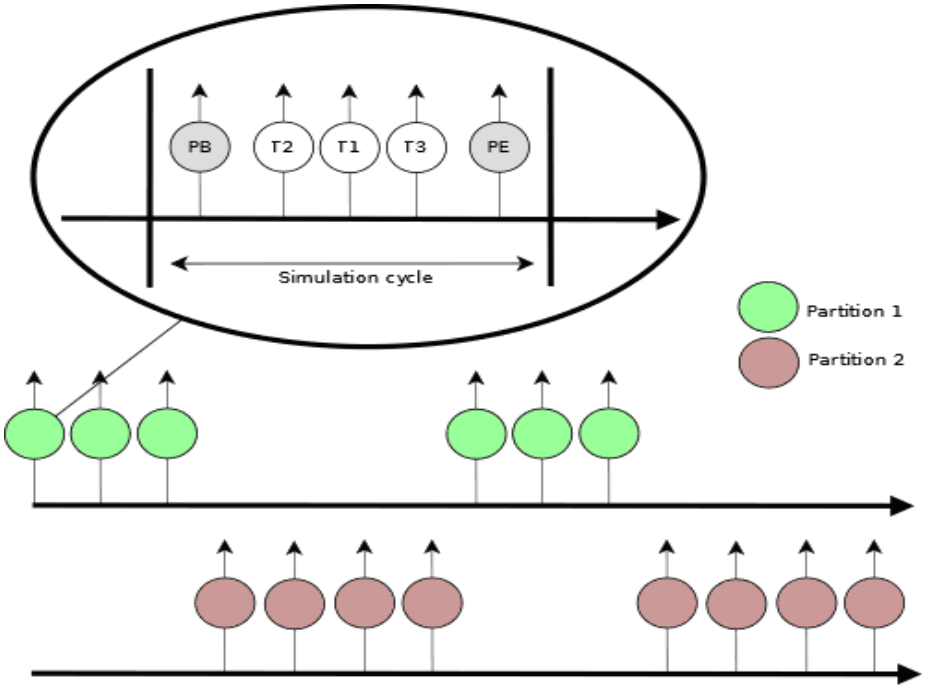

Fig. 4. Simulation of a multi-partition system

2.4 About Determinism in Marzhin.

Despites the intrinsic randomness of the Marzhin simulator, a deterministic behavior is observed most of the times, thanks to the rigorous management of the THREAD priorities. However, in some situations, it becomes possible to introduce a certain level of non-determinism that can be useful for analysis purposes.

In the example below, randomness occurs with a Rate Monotonic scheduler when several threads have the same period and therefore have the same priority:

Simulation configuration:

```
process1 : RATE_MONOTONIC_PROTOCOL
thread1  : DispatchProtocol=PERIODIC Period=10 WCET=3
thread2  : DispatchProtocol=PERIODIC Period=10 WCET=3
thread3  : DispatchProtocol=PERIODIC Period=10 WCET=3
```

Simulation trace:

```
THREAD process1.thread3  __|_|___|._||__|....|__|____|.
THREAD process1.thread2  _|_|_|....|_____||..._|____||..
THREAD process1.thread1  |_____||..___||___|.__|_||....
```

Caption:

```
. : THREAD_STATE_SUSPENDED
| : THREAD_STATE_RUNNING
_ : THREAD_STATE_READY
```

During the simulation cycle 0, the random routine selected thread1 whereas it is thread2 in cycle 1, and so on. It is however possible to control this non-determinism thanks to the Quantum and Dispatch_Order attributes. Quantum specifies the minimum amount of time the currently selected THREAD will remain active without being prempted and Dispatch_Order indicates how the current THREAD is selected within the list (FIRST, LAST or RANDOM). The same example with a Quantum set at 3 and a Dispatch_Order set at FIRST gives the following simulation trace:

```
THREAD process1.thread3  ______|||._____|||._____
THREAD process1.thread2  ___|||...___|||...___||
THREAD process1.thread1  |||.......|||.......|||..
```

The non-determinism of Marzhin can also be beneficial to manage the Global Asynchronism of the simulation environment. It is thus possible to inject events or update data values in incoming ports connected to remote input devices such as the operator keyboard, a dedicated dialog box or an active widget in a 3D virtual reality simulation.

The following example shows how an event can dynamically influence the behavior of the simulation. The periodic THREAD thread1 sends an event to the sporadic THREAD thread2. Such an event could also come from external interface of the simulator:

```
process1 : RATE_MONOTONIC_PROTOCOL
thread1  : DispatchProtocol=PERIODIC Period=10 WCET=5
thread2  : DispatchProtocol=SPORADIC Period=4  WCET=3

EVT IN process1.thread2.evt .....................11......
THREAD process1.thread2     .....................|||....
THREAD process1.thread1     .|||||.....|||||.....|___||||
```

Caption:

```
1 : number of events in the incoming port queue.
```

3 Virtual Execution of AADL Models

3.1 AADL Inspector

AADL Inspector is a model processing framework composed of an AADL toolbox and a customizable set of plugins. The AADL toolbox includes an AADL parser and the LMP (Logic Model Processing) model processing environment [4] that is based

on the use of the prolog language. The LMP engine is used to perform queries on the AADL declarative and instance models, to implement static model checkers and to develop model transformations.

For Real Time analysis, two plugins are currently embedded in AADL Inspector: Cheddar [1] that implements feasibility tests and a static simulator, and Marzhin for dynamic simulation. The static simulator graphically reflects the deterministic outcome of the scheduling analysis, whereas the dynamic simulator exhibits the behavior of the multi-agent engine execution. The result of both simulators is displayed graphically in an advanced time lines viewer.

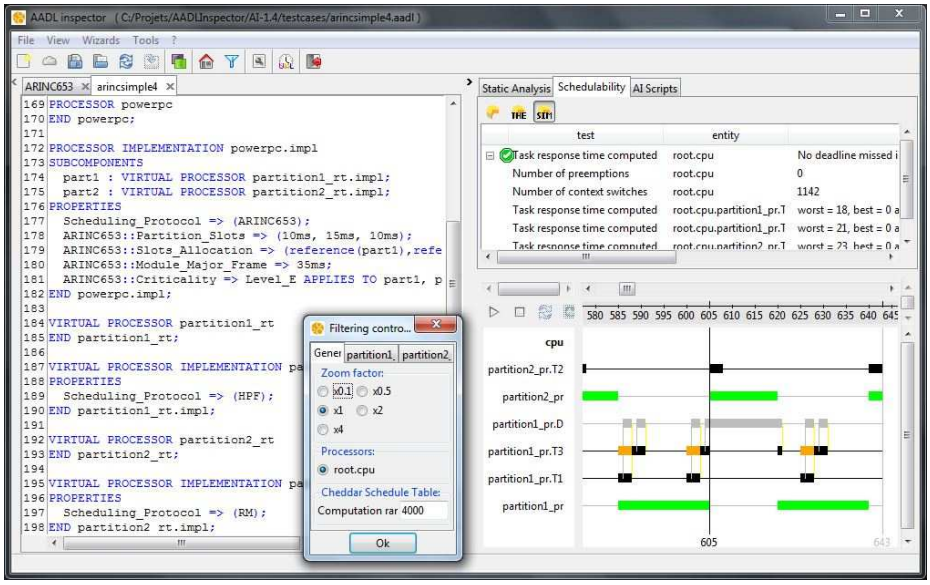

Fig. 5. AADL Inspector 1.4

Thanks to AADL Inspector, it is thus possible to load a complete AADL project distributed on several files containing textual declarative statements, to analyse it in order to build the corresponding instance model, to perform the proper model transformation so that it can be processed by Marzhin, and to pilot its virtual execution through a control panel.

3.2 Executing AADL models

Such a virtual execution of AADL models can efficiently complements the use of more formal real time analysis tools such as Cheddar, as it does not require the input model to satisfy restricted assumptions. It thus significantly extends the scope of model driven real time analysis, especially in the direction of non-periodic activities.

Another use of virtual execution is to perform architecture trade-off studies by providing an immediate feedback showing the coarse grain dynamic behavior of the system during the design phases.

Finally, the specific technical approach that has been chosen for the implementation of Marzhin enables an easy interaction with an asynchronous environment, such as a human operator or a virtual reality simulation.

This approach can be operated early in the development process of the system to support system and software real-time design activities, before the software coding phases. Although the AADL Behavior Annex is used to describe the concurrent aspects of the system behavior, purely procedural actions are still expressed by their computation time. Further work would be required to investigate the ways to enrich this approach with automatic code generation capabilities.

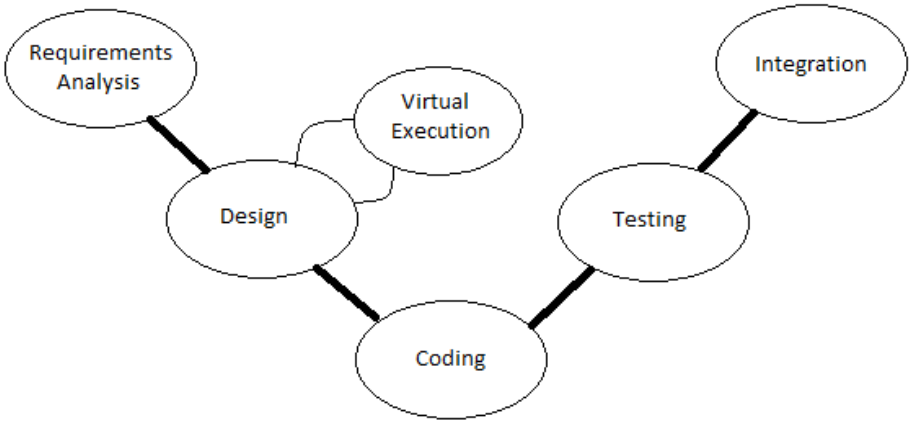

Fig. 6. Use of Virtual Execution in the development life cycle

Conclusion and Future Work

The current implementation of the Marzhin simulator that is available as a part of the AADL Inspector tool already supports a comprehensive subset of the AADL runtime semantics that enables virtual execution of models for the purpose of Real Time analysis, exploration of design solutions and early demonstration of the behavior of a future system.

This work is partly realized in the context of the SMART project [3] in collaboration with the University of Brest and with the financial support of the Council of Brittany, the Council of Finistère, BMO and BPI France.

The future improvements that are foreseen for this activity concern a more complete implementation of the AADL Behavior Annex, improved support of distributed systems and investigations around the possible benefit of the approach for system safety analysis with a proper use of the AADL Error Annex. An additional topic could be studying the possible implications for automatic code generation.

References

1. F. Singhoff, J. Legrand, L. Nana, L. Marcé. “Cheddar: a Flexible Real-Time Scheduling Framework”, ACM SIGAda Ada Letters, 24(4):1-8, ACM Press. 2004
2. SAE International. “Architecture Analysis and Design Language (AADL)”, AS5506B. 2012
3. P. Dissaux, O. Marc, S. Rubini , C. Fotsing, V. Gaudel, F. Singhoff, A. Plantec, Vương Nguyễn-Hồng, Hải Nam Trần. “The SMART Project: Multi-Agent Scheduling Simulation of Real-time Architectures”, Proceedings ERTS conference. 2014.
4. P. Dissaux, P. Farail. “Model Verification: Return of experience”, Proceedings ERTS conference. 2014.
5. Ellidiss Technologies. AADL Inspector site: http://www.ellidiss.fr