AADL Modeling and Analysis of Hierarchical Schedulers

Frank Singhoff, Alain Plantec
LISYC/EA 3883, University of Brest
20, av Le Gorgeu
CS 93837, 29238 Brest Cedex 3, France
{singhoff,plantec}@univ-brest.fr

ABSTRACT

A system based on a hierarchical scheduler is a system in which the processor is shared between several collaborative schedulers. Such schedulers exist since 1960 and they are becoming more and more investigated and proposed in real-life applications. For example, the ARINC 653 international standard which defines an Ada interface for avionic real time operating systems provides such a kind of collaborative schedulers. This article focuses on the modeling and the performance analysis of hierarchical schedulers. We investigate the modeling of hierarchical schedulers with AADL. Hierarchical scheduler timing and synchronization relationships are expressed with a domain specific language based on timed automata: the Cheddar language. With the meta CASE tool Platypus, we generate Ada packages implementing the Cheddar language. These Ada packages are part of Cheddar, a real time scheduling simulator. With these Ada packages, Cheddar is able to perform analysis by scheduling simulation of AADL systems composed of hierarchical schedulers. An AADL model of the ARINC 653 hierarchical scheduling is described as an illustration.

Keywords

AADL, Cheddar, Platypus, Ada framework, Real time scheduling analysis, Timed Automaton.

General Terms

Performance, Reliability, Verification.

Categories and Subject Descriptors

SOFTWARE ENGINEERING [**Software/Program Verification**]: Validation

1. INTRODUCTION

In [32], we presented Cheddar, a set of Ada packages which aims at performing analysis of real time applications. This set of packages includes analytical scheduling tools and most of classical scheduling simulation algorithms. It also provides a domain specific language called the Cheddar language. The Cheddar language allows the designer to define new schedulers which are not already implemented into the Cheddar framework. The Cheddar framework offers a set of tools (eg. interpreters) for such user-defined schedulers.

Cheddar is able to perform analysis on AADL specifications. The SAE Architecture Analysis and Design Language (AADL) is a textual and graphical language support for model-based engineering of embedded real time systems that has been approved and published as SAE Standard AS-5506 [14]. AADL is used to design and analyze the software and the hardware architecture of embedded real-time systems.

In [33], we explained how performance analysis can be performed with Cheddar on AADL specifications when threads are scheduled with usual schedulers such as Rate Monotonic or Earliest Deadline First. This article presents the support of hierarchical schedulers with AADL and Cheddar. A system based on a hierarchical scheduler is a system in which the processor is shared between several collaborative schedulers. With the current AADL standard and with the current Cheddar implementation, such a scheduler is difficult to model and analyze.

Hierarchical scheduling has been initially proposed in the context of time sharing systems. In time sharing systems, hierarchical schedulers were proposed in order to define user-level scheduling policies (eg. fair process scheduling [20] or user-level and kernel-level threads scheduling into Solaris [36]). If user-level scheduling capability stays a motivation for the use of hierarchical schedulers, system designers mostly focus on hierarchical scheduling in order to reduce system design cost and to increase the sharing resource efficiency. Today, it is usual to share a processor by several applications. This allows old applications to run efficiently on newer processors without beeing re-designed (eg. re-design of the scheduling). Applications sharing a processor can have different resource requirements. For example, in a real time multimedia application, a given scheduler may support critical tasks for audio and video presentation while uncritical tasks can be managed by a different scheduler which does not provide deterministic task response time. The queueing based hierarchical scheduling features proposed by POSIX 1003 or Ada 2005 allow such a differentiated class resource allocation [28, 26, 29, 7, 16]

Unfortunately, real time hierarchical schedulers also raise two difficult challenges.

1) The first challenge is related to the large number of hier-

SIGAda'07, November 4–9, 2007, Fairfax, Virginia, USA.

archical schedulers which were proposed. These hierarchical schedulers have complex and different ways to perform communication and synchronization relationships between the schedulers and the tasks [4]. It is also difficult to express scheduler requirements and behaviors in order to combine themself for example [21, 27]. In contrary to the usual schedulers such as Earliest Deadline First or Rate Monotonic, it is difficult to implement into the Cheddar framework a set of hierarchical schedulers satisfying most of system designer needs. In the context of hierarchical schedulers, the approach proposed by Cheddar is to provide a programming language which eases the design and the implementation of hierarchical schedulers.

2) The second challenge is related to the availability of analytical methods for hierarchical schedulers: there is currently few feasibility tests in the context of real time hierarchical scheduling [1, 12, 30, 10]. A feasibility test is an analytical method which is able to predict before task execution if a given system will meet its task timing requirements. Building feasibility tests is usually a difficult work and it is more complex for hierarchical schedulers. When no feasibility test exists for a given hierarchical scheduler, Cheddar can be used to perform scheduling simulation. In this case, the scheduling simulation tool has to be efficient (low memory footprint and high response time) in order to run large simulations.

The Cheddar language was formerly introduced in [32]. This language made it possible the design of user-defined schedulers with fixed timing and synchronization relationships between tasks and schedulers. A program written with the Cheddar language is organized in sections. A section is a kind of Ada sub-program. The former Cheddar scheduling simulation engine assumed a fixed order for the execution of such sections: this fixed order was modeling the fixed timing/synchronization relationships between tasks and schedulers.

In this article, we propose to extend the Cheddar language and its tools (editor, interpreter and compiler) in order to design hierarchical schedulers with AADL. This new Cheddar language allows the designer to model any synchronization and timing relationships between the tasks and the schedulers. Tasks and schedulers relationships are modeled with timed automata [13, 3]. The abstract semantic of the Cheddar language is modeled with the meta CASE tool Platypus. From this Cheddar language model, Platypus is able to generate Ada packages which implement tools such as Cheddar program compiler or interpreter.

Timed automata are frequently used to express timing and synchronization requirements of real time systems. There is some experiments to model and verify real time schedulers with timed automata [2, 35, 19]. Numerous tools exist (editors, simulators and model-checkers such as UPPAAL [5] or Esterel Studio [6]) and some standards are also based on such a formal model (eg. UML Statecharts [11]).

Timed automaton is also the formal model chosen by the SAE AADL standard committee to express AADL behavioral properties in the next release of the AADL standard [15] and first experiments on the verification of AADL specifications with timed automata were presented recently [8]. By extending the Cheddar language with a timed automaton model similar to the one investigated by the SAE AADL committee, the work presented in this article is then a first contribution to the scheduling analysis of AADL specifica-

```
thread T1
end T1;

thread implementation T1.Impl
   properties
      Dispatch_Protocol => Periodic;
      Compute_Execution_Time => 3 ms .. 3 ms;
      Cheddar_Properties::Fixed_Priority => 2;
      Deadline => 10 ms;
      Period => 10 ms;
end T1.Impl;
...
process implementation partition1.Impl
   subcomponents
      T3 : thread T3.Impl;
      T4 : thread T4.Impl;
   properties
      Cheddar_Properties::Scheduling_Protocol
         => Automaton_User_Defined_Protocol;
      Cheddar_Properties::Source_Text
         => "arinc_partition1.sc";
      Cheddar_Properties::Automaton_Name
         => "partition1_scheduler";
end partition1.Impl;

process implementation partition2.Impl
   subcomponents
      T1 : thread T1.Impl;
      T2 : thread T2.Impl;
   properties
      Cheddar_Properties::Scheduling_Protocol
         => Automaton_User_Defined_Protocol;
      Cheddar_Properties::Source_Text
         => "arinc_partition2.sc";
      Cheddar_Properties::Automaton_Name
         => "partition2_scheduler";
end partition2.Impl;

processor implementation arinc.Impl
   properties
      Scheduling_Protocol
         => Automaton_User_Defined_Protocol;
      Cheddar_Properties::Source_Text
         => "arinc_processor.sc";
      Cheddar_Properties::Automaton_Name
         => "processor_scheduler";
end arinc.Impl;
...
system auto_arinc
end auto_arinc;

system implementation auto_arinc.Impl
   subcomponents
      arinc : processor arinc.Impl;
      partition1 : process partition1.Impl;
      partition2 : process partition2.Impl;
   properties
      Actual_Processor_Binding => reference
         arinc applies to partition1;
      Actual_Processor_Binding => reference
         arinc applies to partition2;
end auto_arinc.Impl;
```

Figure 1: A part of an AADL specification modeling an ARINC 653 avionic system

tions using AADL behavioral features [15].

This article is organized as follows. The sections 2 and 3 are devoted to the AADL modeling of hierarchical scheduler based systems. The section 2 focuses on the architecture point of view while the section 3 focuses on the timing and the synchronization point of view. In section 4, we describe what kind of analysis the AADL designer can expect with Cheddar on such AADL specifications. We also explain how Ada packages implementing analysis tools are generated from an AADL specification. Finally, we conclude and give future works in section 5.

2. MODELING HIERARCHICAL SCHEDULERS WITH AADL

An AADL specification describes both the hardware part and the software part of a real time system [14]. An AADL specification is a set of components such as shared data, threads, processes (the software side of a specification), processors, devices and busses (the hardware side of a specification).

In the sequel, we focus on thread, process and processor components. A thread is a sequential flow of control that executes a program. An AADL thread may be implemented by an Ada task. AADL threads can be woken up according to several policies: a thread may be periodic, sporadic or aperiodic. An AADL periodic thread is woken up at a regular time interval. This time interval is called a "period". In the case of a sporadic thread, a minimum inter-woken up time interval is considered. An aperiodic thread may be woken up at any time. An AADL process models a virtual address space. In the most simple case, a process simply owns threads and shared data. Finally, a processor is the execution platform component which is capable of scheduling and executing threads.

An AADL specification also contains component connections and component properties. Component connections model component relationhips such as thread precedency constraints, message exchanges or shared data access. Component properties store component information which is related to the component behavior or the way the component will be implemented in the target system. A property is defined by a name, a value and a type. For example, a thread component property may store the Ada package file name in which the Ada task implementing the AADL thread will be defined. The designer can define properties with most of the AADL components. If AADL defines standard properties, AADL also allows the designer to define its own properties.

Figure 1 shows a part of an AADL specification modeling a simple ARINC 653 avionic system. ARINC 653 provides space and time partitioning when several applications share the same processor and the same memory unit. An ARINC 653 system is then a set of applications called partitions. Each partition is composed of tasks. The processor sharing is made according to a two-levels hierarchical scheduling:

1. The partitions are cyclically activated. This first level of scheduling is fixed at design time: it is usually statically computed.

2. The second scheduling level is related to the task scheduling: tasks of a given partition are scheduled all together with a fixed priority scheduler. This thread scheduling is an online scheduling.

The ARINC 653 model of the figure 1 specifies a set of AADL periodic threads (threads $T1.Impl$, $T2.Impl$, ..) modeling ARINC 653 tasks. The timing behavior of each thread is defined by a set of properties such as $Period$, $Deadline$ or $Cheddar\_Properties::Fixed\_Priority$. $Period$ and $Deadline$ are AADL standard properties. The $Period$ property stores the fixed delay between each thread wake up time. The $Deadline$ expresses the timing constraint the thread has to meet. Finally, $Cheddar\_Properties::Fixed\_Priority$ is an example of AADL property proposed by Cheddar in order to apply real time scheduling analysis tools. This Cheddar property assigns a fixed priority for each thread of the modeled system. Such a priority attribute is used by fixed schedulers such as Rate Monotonic [23].

The AADL specification of figure 1 also defines two AADL processes which model two ARINC 653 partitions ($partition1.Impl$ and $partition2.Impl$). The $Actual\_Processor\_Binding$ property is a usual AADL way to express that the two processes are run on the same processor: the $arinc.Impl$ processor. With AADL, each processor owns a scheduler. The standard AADL $Scheduling\_Protocol$ property stores the name of the scheduling protocol describing how the processor time is shared between threads. With AADL/Cheddar, when a hierarchical scheduler is modeled, the designer has to provide a $Scheduling\_Protocol$ property for both processes and processors. Since this feature is not AADL V1.0 compliant, a Cheddar specific property is then defined. In this example, for the $arinc$ processor and for the $partition1.Impl$ process, the $Scheduling\_Protocol$ property contains $Automaton\_User\_Defined\_Protocol$ which indicates that the scheduler is a user-defined scheduler modeled by a Cheddar program (eg. a timed automaton). Finally, each timed automaton has a name which is specified by the $Cheddar\_Properties::Automaton\_Name$. In the ARINC example, $partition1\_scheduler$ is the name of the timed automaton modeling the timing/synchronization behavior of the thread scheduler of partition 1 and $processor\_scheduler$ is the automaton name of the scheduler modeling the sharing of the processor time between the two ARINC partitions.

In the next section, we give some details about the timing and the synchronization specification of these schedulers.

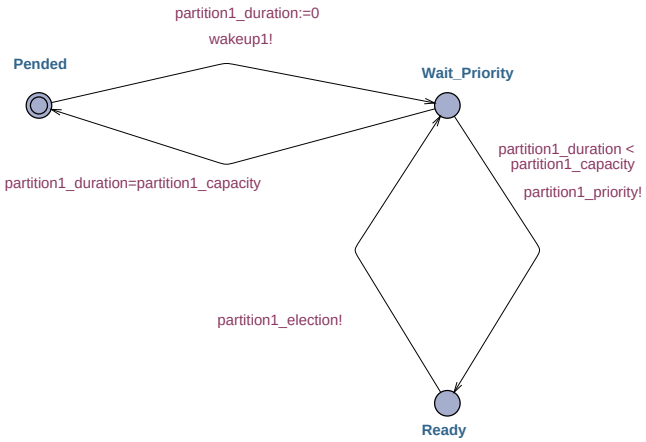

Figure 2: Automaton modeling the thread scheduler of the partition 1

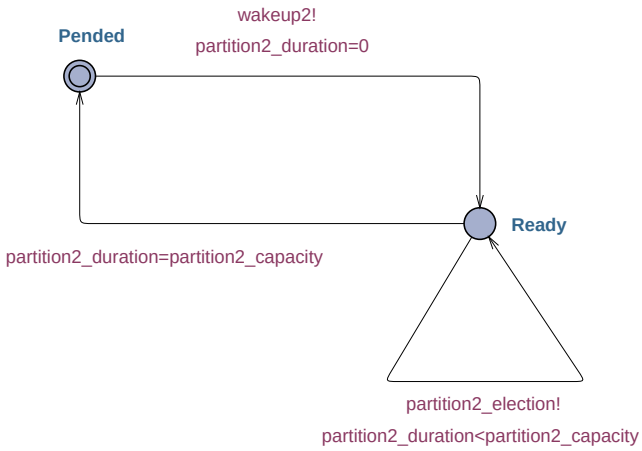

Figure 3: Automaton modeling the thread scheduler of the partition 2

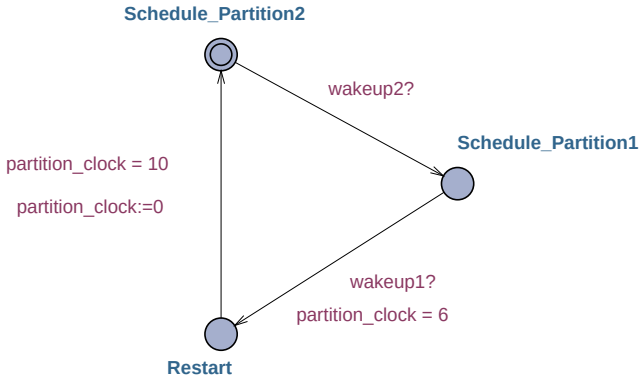

Figure 4: Automaton modeling the ARINC 653 partition scheduler

3. MODELING HIERARCHICAL SCHEDULER BEHAVIOR WITH CHEDDAR

In this section, we give an overview of the Cheddar language. A complete description of the language can be found in [31]. The use of the Cheddar language is then illustrated by the example of the ARINC 653 model of the section 2.

3.1 Outline of the Cheddar language

The Cheddar language is defined by two parts : a small Ada-like language and a timed automaton language.

3.1.1 A small Ada-like language

This small Ada-like language is used to express computations on simulation data. Simulation data are constants or variables used by the scheduling simulator engine of Cheddar in order to run scheduling simulations. Examples of such simulation data are task wake up times or task priorities. A Cheddar program is organized in sections. A section is a kind of Ada sub-program composed of several statements. Most of the time, a Cheddar program is composed of the following sections [32]: a *start_section* which contains variable declarations and initializations ; a *priority_section* which contains the code to compute simulation data on each unit of time during simulation; and an *election_section* which

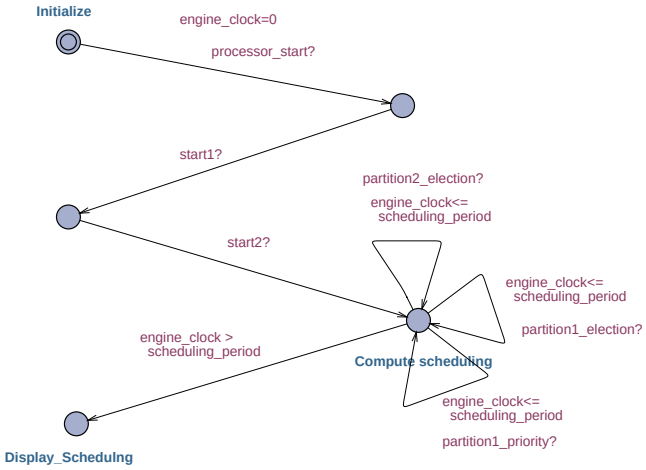

Figure 5: Automaton modeling the Cheddar scheduling simulation engine

looks for the task to run during simulation time. The Ada-like language provides usual types, operators or statements. It also provides statements, types and operators which are more specific to the design and the debug of scheduling algorithms. For example, the $uniform/exponential$ statements customize the way random values are generated during simulation time ; the lcm operator computes last common multiplier of simulation data ; finally, the max_to_index operator looks for the ready task which has the highest value for a given array of simulation data. The detailed Backus-Naur Form syntax of this language is given in [31].

3.1.2 A timed automaton language

A Cheddar program can contains a set of timed automata similar to those proposed by UPPAAL [13, 3, 5]. UPPAAL is a toolbox for the verification of real time systems. A system is then modeled as a network of several timed automata extended with variables. A state of the system is defined by the locations of all automata, the clock constraints, and the values of the variables. Every automaton may fire a transition separately or synchronize with another automaton, which leads to a new state. Transition may be guarded with time constraint. Finally, delays can express time consumption at transition firing. In Cheddar, these timed automata allow to model timing and synchronization behaviors of schedulers. Automata may run Ada-like sections, read or write simulator data.

3.2 Cheddar program examples: an illustration with the hierarchical ARINC scheduler

In section 2, we described with AADL the architectural point of view of an ARINC model made with a set of collaborative schedulers (process schedulers and processor schedulers). This section shows how the Cheddar language can be used to express timing and synchronization relationships between such ARINC partition schedulers and ARINC task schedulers.

The ARINC 653 hierarchical scheduler is modeled by a set of Cheddar programs. A textual representation of those programs is given in the sections 8.1, 8.2 and 8.3. A graphical representation is also given in figures 2, 3, 4.

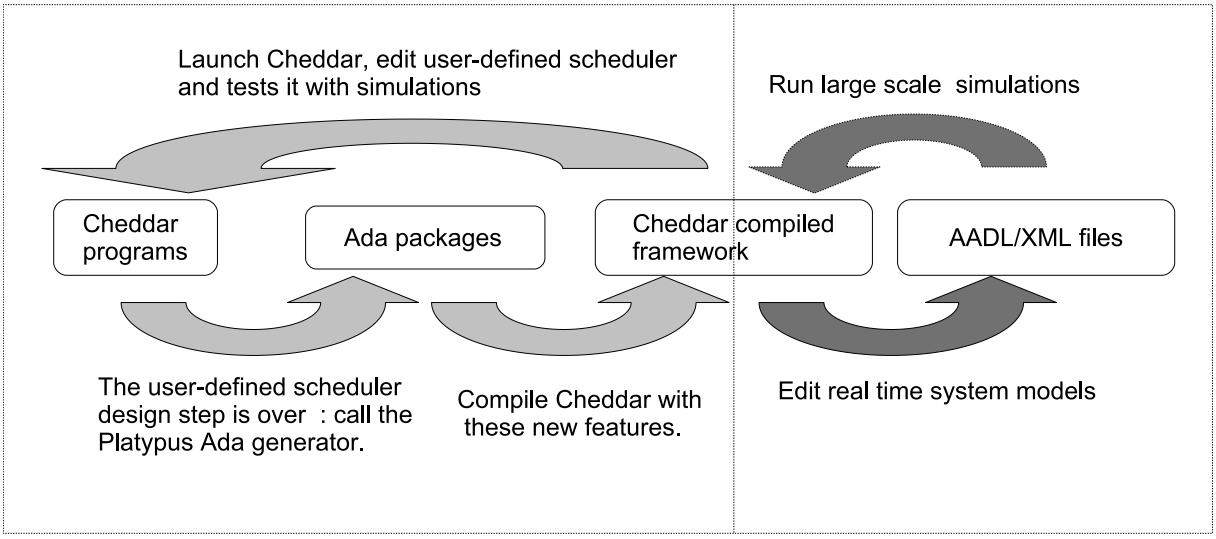

Figure 6: The user-defined scheduler design process with Cheddar

For each AADL process and AADL processor component which has a scheduler, the AADL specification binds a Cheddar program to the component. This binding is made with the $Cheddar\_Properties::Source\_Text$ Cheddar property. The property contains the file name storing the corresponding Cheddar program. For example, in the AADL specification of the figure 1, the automaton in the figure 4 is saved in a file called $arinc\_processor.sc$ and the automaton of the figure 2 is saved in a file called $arinc\_partition1.sc$.

Let see now the modeling of the ARINC 653 thread and partition scheduling. The automaton of the figure 4 specifies when each partition has to be active or not. This automaton models the first level of an ARINC 653 scheduling: the partition scheduling. The automata of the figures 2 and 3 model the schedulers of each partition. Such schedulers are responsible for the scheduling of the threads of the modeled ARINC partition. These automata model the second level of an ARINC 653 scheduling: the task scheduling.

The automata modeling the thread scheduling of each partition have three types of location:

1. the *Pended* locations. From these locations, the partitions can not get access to the processor in order to run one of their thread.

2. the $Wait\_Priority$ and the *Ready* locations. If a partition is in one of these locations, it is allowed to run one of its thread. $Wait\_Priority$ is an intermediate location from which the scheduler computes thread priorities. AADL thread priorities are computed by the $partition1\_priority$ section during the firing of the $Wait\_Priority$ outgoing transition (see the Cheddar program of section 8.1). The *Ready* location chose the AADL thread to run during the next unit of time. To find the next thread to run, the Cheddar program interpreter calls the $partition1\_election$ section during the firing of the *Ready* outgoing transition.

The partition scheduler automaton (see figure 4) models the cyclic partition activation: in this example, the partition scheduling is made on a 10 units of time cycle. Each cycle, the partition 2 is activated during the 6th first units of time and the partition 1 is activated during the 4th last units of time. The partition 2 schedules critical periodic tasks according to Rate Monotonic whereas the partition 1 schedules a set of uncritical tasks according to a round-robin scheduler. The partition scheduler enforces timing isolation between the two partitions which have different processor resource requirements.

The last automaton (see figure 5) models the scheduling engine of Cheddar: this Cheddar program is a part of the Cheddar program interpreter which drives scheduling simulations.

Besides the sections which store timed automaton modeling timing and synchronization behavior, a Cheddar program also contains sections to perform arithmetic/logic statements (eg. to compute task priorities), to do initializations and to select the task to run at the next unit of time.

In the case of our ARINC 653 model, the section $start1$ of the program depicted in section 8.1 does some initializations ; the $partition1\_priority$ of the section 8.1 computes task priorities according to a round robin scheduling with a 2 units of time quantum ; finally, the section $partition2\_election$ of the section 8.2 shows how to select the next highest priority task to run.

4. PERFORMANCE ANALYSIS OF A CHEDDAR PROGRAM MODELING HIERARCHICAL SCHEDULERS

Scheduling simulation consists in predicting for each unit of time, the thread to which the processor should be allocated. Checking if threads meet their deadlines can then be performed by analysis of the computed scheduling. When AADL specifications only contain periodic AADL threads and for some real time schedulers such as Rate Monotonic, scheduling simulations can prove that AADL threads will meet their deadline. For such a proof, the designer has to run the scheduling simulation during a time interval called the scheduling period (sometimes called schedule length, base period, major cycle or hyper period). If the AADL

specification is composed of periodic AADL threads which arrive in the system at the same time, this scheduling period can be computed by [22, 9]:

$$[k, k + 2 * LCM(\forall i : P_i)] \quad (1)$$

where k is the time when all AADL threads request the processor for the first time (eg. thread arrival time), P_i is the period of the thread i and LCM is the last common multiplier of all AADL thread periods of the system. If the system designer run a scheduling simulation from the time k to the time $k+2*LCM(\forall i : P_i)$ and if no thread deadline are missed during such a scheduling period, then, no deadline will be missed during all the thread scheduling.

From a Cheddar program which models a hierarchical scheduler, we generate Ada packages which are part of the Cheddar framework and that allow the designer to run scheduling simulation. Let see now how such Ada packages are generated.

4.1 From Cheddar programs to scheduling simulations

As depicted by figure 6, a new scheduler described by a Cheddar program can be designed and directly interpreted using the Cheddar environment. This feature eases the design step and allows the user to perform small scheduling simulation. But scheduling simulation tools have to be efficient in order to run large simulations. They must have a low memory footprint and a high response time. When the design step is over, the Cheddar program specifying a new scheduler can be handled by a code generator that produces a set of Ada packages. Then, a new Cheddar version that integrates the new scheduler as a builtin one can be compiled. The new Cheddar environment can then run efficient scheduling simulations with the new user-defined scheduler.

The Ada package generator is implemented within Platypus. Platypus [25] is a meta-environment suitable for model driven engineering activities. It allows meta-model specification describing meta-data hierarchies, integrity and transformation rules definition and also allows conforming data models specification. Meta-models as well as conforming models are specified with the EXPRESS modeling language [18]. The STEP technology [17] is used in order to automatically instanciate meta-models from their conforming models. Then, transformation rules associated to meta-entities can be interpreted in order to generate some target realization such as an Ada package for example.

From a Cheddar program, Platypus generates two different Ada packages: the Ada packages implementing the modeled scheduler and the Ada packages which are part of the Cheddar data access interface ($CDAI$) [24, 34].

4.2 The Cheddar data access interface

As shown by figure 7, the CDAI is a central component of Cheddar. It implements a repository that all other components are using in order to read and write data or meta-data. Mainly, Cheddar user interface and builtin scheduler components are using it in order to get or put simulation data such as processor or task attributes. The Cheddar language compiler and interpreter are using it in order to store Cheddar programs meta-data and to manage computation results.

4.3 The Cheddar program interpreter

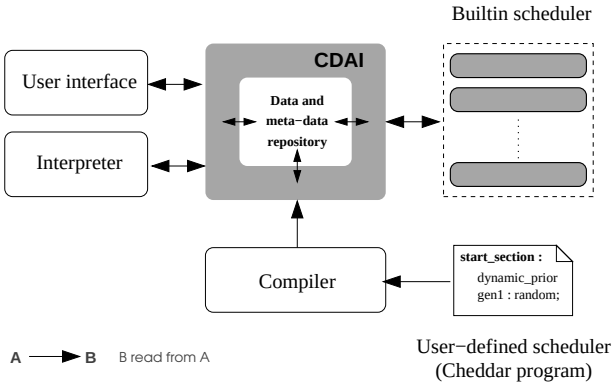

Figure 7: The Cheddar data acces interface

The Cheddar program interpreter allows the designer to test Cheddar program. During a scheduling simulation, the Cheddar program interpreter is called at each unit of time in order to evaluate automaton transitions which can be fired. Transition firing may lead to run priority or election sections. Start sections are always run by the simulation engine at simulation start time.

During scheduling simulation, the interpreter maintains a status for each transition of the Cheddar programs. These status can be:

- *Guarded*: the transition can not be fired due to a transition guard which is not satisfied.
- *Pended*: the transition is executing a time consumming statement and then, can not be fired until the delay is not elapsed.
- *Unreachable*: the transition is not allowed to be fired according to the current location of the automaton.
- *RendezVous*: the transition can not be fired because the automaton is waiting for synchronization with another automaton.
- *Ready*: the transition is ready to be fired.

For a given nth unit of time, the Cheddar program interpreter works as follows:

1. It checks and updates the status of each transition:
   - A transition which is "pended" stays "pended" if the delay is not exhausted. If its delay is exhausted, the automaton status becomes "ready".
   - For all others transitions, their status is set to "ready" immediately.
2. It checks reachability of all "ready" transitions. The transition status is set to "unreachable" if the transition is not an outgoing transition of the current location of the corresponding automaton.
3. It checks guard (eg. timed constraints) for all transitions which have a "ready" status. A transition blocked due to a guard constraint sees its status becoming "guarded".

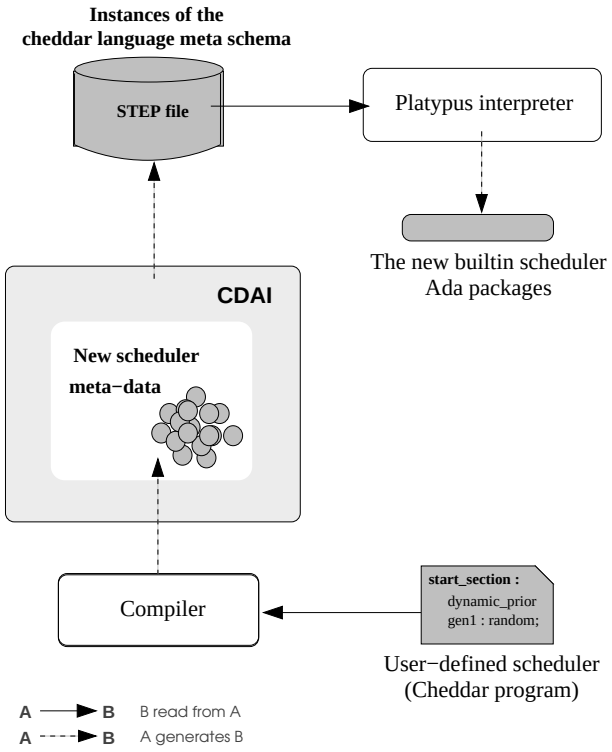

Figure 8: The user-defined scheduler generation process

4. It checks synchronization constraints for all transitions which have a "ready" status. A transition blocked due to a synchronization constraint sees its status becoming "rendezvous".

5. Finally, it fires all transitions which have a "ready" transition status. Firing a transition means:

   5.1 To run delay/clock statement. This may lead to compute the next wakeup automaton date and to change its status to "pended".

   5.2 To run sections. Section run there can be *priority* or *election* sections. If an election section is run, then, the interperter has ended its work for the current unit of time and must switch to the n+1th unit of time.

   5.3 And then, to update the current location (transition outgoing location) for all fired transitions except for those which became "pended" at step 5.1.

6. The interpreter restarts at step 1 if at least one transition was fired at step 5. Otherwise, the interperter has ended its work for the current unit of time and must switch to the $n + 1$th unit of time.

4.4 Process to generate Ada packages from a Cheddar program

After a Cheddar program was tested by the interpreter, one can use an Ada code generator in order to compile and integrate the new scheduler into the Cheddar framework. This new compiled Cheddar framework makes it possible to efficiently run time consumming simulations (eg. simulation on large model).

This Ada package generation process is depicted by figure 8.

The Cheddar program Ada package generator is made of a meta-model named *cheddar_language_meta_schema* specifying the Cheddar language abstract semantic. It also contains translation rules specifying how to generate Ada packages implementing a scheduler modeled by a Cheddar program.

From the Cheddar language meta-model, the CDAI has been enriched with a dedicated component automatically generated and aiming at Cheddar program metadata handling. Within the repository, Cheddar programs are stored as *cheddar_language_meta_schema* instances produced by the Cheddar program compiler. In order to generate a user-defined scheduler, Platypus is able to read *cheddar_language_meta_schema* instances generated by the CDAI and stored as a STEP exchange file and finally, to generate a new scheduler as a dedicated Ada package.

5. CONCLUSION AND FUTURE WORKS

In this article, we have investigated the modeling and the analysis of AADL specifications containing hierarchical schedulers. We have proposed an extension of the Cheddar language and its tools (editor, interpreter and compiler) in order to support hierarchical schedulers. This new Cheddar language makes use of timed automata [13, 3] and allows the designer to model synchronization and timing relationships between the AADL threads, the schedulers of AADL processors and the schedulers of AADL processes. With the meta CASE tool Platypus, we have designed a meta-model of Ada 95 for Cheddar and a model of the Cheddar language [24, 34]. From these models, we generate Ada packages which are part of the Cheddar scheduling simulation engine. These Ada packages implement a Cheddar program compiler and interpreter. Scheduling simulation analysis can then be performed on AADL specifications with hierarchical schedulers.

An AADL model of the ARINC 653 hierarchical scheduler is described as an illustration.

The SAE AADL committee is currently preparing the next AADL standard release (AADL release V2.0). Such a new release should propose new features related to ARINC 653. The next AADL release should also propose a timed automaton language in order to express behavioral properties of AADL systems [15]. By extending the Cheddar language with a timed automaton model similar to the one investigated by the SAE AADL committee, the work presented in this article is then a first contribution to the scheduling analysis of AADL specifications using AADL behavioral features [15]. In the next months, we will have to check that a Cheddar program can be actually transformed towards an AADL V2.0 behavioral specification.

6. ACKNOWLEDGMENTS

Cheddar is an open-source tool (available at http://beru.univ-brest.fr/~singhoff/cheddar/) and many people help the Cheddar team by their advices, bug reports or documentation/source code contributions. The Cheddar team would like to thank all contributors. The list of the contributors can be reached at http://beru.univ-brest.fr/~singhoff/cheddar/#Ref7. Cheddar AADL analysis features rely on Ocarina. We also would like to thank the

ENST Ocarina's Team (B. Zalila, J. Hugues, L. Pautet and F. Kordon).

7. REFERENCES

[1] L. Almeida and P. Pedreiras. Scheduling within Temporal Partitions : response-time analysis and server design. *Proceedings of the EMSOFT'04 conference. September 27-29, Pisa, Italy*, pages 95–103, 2004.

[2] K. Altisen, G. Gossler, and J. Sifakis. Scheduler Modeling Based on the Controller Synthesis Paradigm. *Real Time Systems journal*, 23(1):55–84, 2002.

[3] R. Alur and D. L. Dill. Automata for modeling real time systems. Proc. of Int. Colloquium on Algorithms, Languages and Programming, Vol 443 of LNCS (1990) 322–335, 1990.

[4] T. E. Anderson, B. N. Bershad, E. D. Lazowska, and H. M. Levy. Scheduler activations: Effective kernel support for the user-level management of parallelism. *ACM Transactions on Computer Systems*, 10(1):53–79, February 1992.

[5] G. Behrmann, A. David, and K. G. Larsen. A Tutorial on UPPAAL. Technical Report Updated the 17th November 2004, Department of Computer Science, Aalbord University, Denmark, 2004.

[6] G. Berry. Getting Started with Esterel Studio 5.3. Technical report, Esterel technologies SA. Available from http://www.esterel-technologies.com/technology/getting-started/, April 2005.

[7] A. Burns, M. Harbour, and A. Wellings. A round robin scheduling policy for Ada. *In Reliable Software Technologies, Proceedings of the Ada Europe Conference*, 2003.

[8] D. Chemouil and N. Pontisso. Vérification formelle d'un modèle AADL à l'aide de l'outil UPPAAL. *Revue Génie Logiciel*, (80):36–40, March 2007.

[9] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. *Scheduling in Real Time Systems*. John Wiley and Sons Ltd editors, 2002.

[10] R. I. Davis and A. Burns. Hierarchical Fixed Priority Pre-Emptive Scheduling. *In the 26th IEEE International Real-Time Systems Symposium (RTSS'05). Miami, Florida, USA.*, pages 389–398, December 2005.

[11] D. Drusinsky. *Modeling and Verification using UML StateCharts*. Elsevier inc. editor, 2006.

[12] M. G. Harbour and J.C. Palencia. Response Time Analysis for Tasks Scheduled under EDF within Fixed Priorities. In *Proceedings of the 24th IEEE Real-Time Systems Symposium, Cancun, Mexico*, December 2003.

[13] J. E. Hopcroft and J. D. Ullman. Introduction of Automata Theory, Languages and Computation. Addison-Wesley editor, 2001.

[14] SAE Inc. Architecture Analysis and Design Language (AADL) AS 5506. Technical report, The Engineering Society For Advancing Mobility Land Sea Air and Space, Aerospace Information Report, Version 0.994, August 2004.

[15] SAE Inc. AADL Annex Behavior (draft V1.6), AS 5506. Technical report, The Engineering Society For Advancing Mobility Land Sea Air and Space, Aerospace Information Report, March 2007.

[16] ISO. Ada Reference Manual ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1 (Draft 16).

[17] ISO 10303-1. *Part 1: Overview and fundamental principles*, 1994.

[18] ISO 10303-11. *Part 11: EXPRESS Language Reference Manual*, 1994.

[19] T. K. Iversen, K. J. Kristoffersen, K. G. Larsen, R. G. Madsen, M. Laursen, S. K. Mortensen, P. Pettersson, and C. B. Thomasen. Model-Checking Real Time Control Programs : Verifying LEGO Mindstorm Systems Using UPPAAL. Technical report, BRICS RS-99-53, December 1999.

[20] J. Kay and P. Lauder. A Fair Share Scheduler. In *Communications of the ACM*, volume 31, pages 44–45, January 1988.

[21] J.L. Lawall, G. Muller, and H. Duchesne. Language Design for implementing Process Scheduling Hierarchies. *Proceedings of the PEPM'04 conferences. August 24-26, Veronna Italy*, pages 80–90, 2004.

[22] J.Y.T Leung and M.L. Merril. A note on preemptive scheduling of periodic real time tasks. *Information processing Letters*, 3(11):115–118, 1980.

[23] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[24] A. Plantec and F. Singhoff. Refactoring of an Ada 95 Library with a Meta CASE Tool. Proceedings of the International ACM SIGAda Conference, Albuquerque , USA, November 2006.

[25] Platypus Technical Summary and download. http://cassoulet.univ-brest.fr/mme/.

[26] J. A. Pulido, S. Uruena, J. Zamorano, T. Vardanega, and J. A. De la Puente. Hierarchical Scheduling with Ada 2005. *Proceedings of the 11th International Conference on Reliable Software Technologies, Porto, Portugal*, June 2006.

[27] J. Regehr and J. A. Stankovic. HLS : a Framework for Composing Soft Real-Time Schedulers. *In the 22th IEEE International Real-Time Systems Symposium (RTSS'01). London, UK.*, pages 3–14, December 2001.

[28] M. Rivas and M. G. Harbour. POSIX-compatible application-defined scheduling in MaRTE OS. *In Proceedings of the 14th IEEE Euromicro Conference on Real-Time Systems, Wien, Austria*, June 2002.

[29] M. Rivas and M. G. Harbour. Application Defined Scheduling in Ada. *Proceedings of the 12th international workshop on Real-time Ada. Viana do Castelo, Portugal.*, pages 42–51, 2003.

[30] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *4th IEEE International Real-Time Systems Symposium (RTSS'03)*, 2003.

[31] F. Singhoff. Cheddar Release 2.0 User's Guide. Technical report, number singhoff-01-2007, Available at http://beru.univ-brest.fr/~singhoff/cheddar, February 2007.

[32] F. Singhoff, J. Legrand, L. Nana, and L. Marcé.

Cheddar : a Flexible Real Time Scheduling Framework. Proceedings of the International ACM SIGAda Conference, Atlanta, USA, November 2004.

[33] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Scheduling and Memory requirements analysis with AADL. Proceedings of the International ACM SIGAda Conference, Atlanta, USA, November 2005.

[34] F. Singhoff and A. Plantec. Towards User-Level extensibility of an Ada library : an experiment with Cheddar. Proceedings of the 12th International Conference on Reliable Software Technologies, Ada-Europe. Geneva, LNCS springer-Verlag, June 2007.

[35] V. Subraminian, C. Gill, C. Sanchez, and H. B. Sipma. Reusable Models for Timing and Liveness Analysis of Middleware for Distributed Real-Time and Embedded systems. Proceedings of the 6th ACM and IEEE International conference on Embedded software EMSOFT '06, October 2006.

[36] U. Vahalia. *UNIX Internals : the new frontiers.* Prentice Hall, 1996.

8. ANNEX

8.1 Cheddar program modeling the partition 1 scheduler

```
start_section  start1 :
  partition1_capacity : integer := 4;
  partition1_duration : clock := 0;
  quantum : integer :=2;
  my_prio : array (tasks_range) of integer;
  my_prio:=0;
end section;

priority_section partition1_priority :
   quantum:=quantum-1;
   if quantum = 0
      then quantum:=2;
           my_prio(previously_elected):=
             my_prio(previously_elected)+1;
   end if;
end section;

election_section partition1_election :
   return min_to_index(my_prio);
end section;

automaton_section  partition1_scheduler  :
  Pended : initial_state;
  Ready : state;
  Wait_Priority : state;

  transition Pended ==>
     [ , partition1_duration:=0;,wakeup1!]
             ==> Wait_Priority;

  transition Wait_Priority ==>
     [partition1_duration<partition1_capacity,
         , partition1_priority!]
             ==> Ready;

  transition Ready ==>
     [ , , partition1_election!]
             ==> Wait_Priority;

  transition Wait_Priority ==>
     [partition1_duration=partition1_capacity , , ]
             ==> Pended;
end section;
```

8.2 Cheddar program modeling the partition 2 scheduler

```
start_section  start2 :
   partition2_capacity : integer := 6;
   partition2_duration : clock := 0;
end section;

election_section  partition2_election :
   return min_to_index(tasks.priority);
end section;

automaton_section  partition2_scheduler  :
  Ready : state;
  Pended : initial_state;

  transition Pended ==>
     [ , partition2_duration:=0; ,wakeup2!]
             ==> Ready;

  transition Ready ==>
     [ partition2_duration < partition2_capacity,
        , partition2_election!]
             ==> Ready;

  transition Ready ==>
     [partition2_duration=partition2_capacity, , ]
             ==> Pended;
end section;
```

8.3 Cheddar program modeling the partition scheduler

```
start_section  start_processor :
      partition_clock : clock:=0;
end section;

automaton_section  processor_scheduler :
  Schedule_Partition2 : initial_state;
  Schedule_Partition1 : state;
  Restart : state;

  transition Schedule_Partition2 ==>
     [ , , wakeup2?]
             ==> Schedule_Partition1;

  transition Schedule_Partition1 ==>
     [partition_clock = 6 , , wakeup1?]
             ==> Restart;

  transition Restart ==>
     [partition_clock = 10 ,  partition_clock:=0; ,]
             ==> Schedule_Partition2;
end section;
```