Scheduling and Memory requirements analysis with AADL

F. Singhoff, J. Legrand, L. Nana, L. Marcé
LISYC/EA 3883, University of Brest
20, av Le Gorgeu
CS 93837, 29238 Brest Cedex 3, France
{singhoff,jlegrand,nana,marce}@univ-brest.fr

ABSTRACT

This article describes an Ada set of packages which allows designers to perform resource requirements analysis of AADL specifications. This set of Ada packages is part of Cheddar, an Ada framework that we are developing at the University of Brest [22].

The framework provides tools to check if AADL threads will meet their deadline at execution time. Some new AADL properties are proposed to model and analyze dependent AADL thread sets. It also provides some tools to perform memory requirements analysis on AADL specifications.

Keywords

AADL, Ada framework, memory requirements and real time scheduling analysis.

General Terms

Performance, Reliability, Verification.

Categories and Subject Descriptors

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

1. INTRODUCTION

In [22], we presented a set of Ada packages which aims at performing performance analysis of concurrent real time applications. This set of packages includes most of classical scheduling simulation methods and classical scheduling feasibility tests in the case of dependent and independent tasks running on monoprocessor and distributed systems. This article presents how this set of packages can be used to perform this kind of analysis on real time applications designed with the help of 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[9]. It is used to design and analyze the software and the hardware architecture of embedded real-time systems. Properties that are critical to the operation of such a system are timing, throughput, and reliability properties. As shown by the studies led by Axlog[1], the SAE AADL is applicable to any performance-critical embedded real-time system in domains such as avionics, aerospace, automotive, and autonomous systems.

A prototype of AADL was previously developed by Honeywell under US Government sponsorship (DARPA and others) to prove the concept. This system called MetaH has been used extensively to validate the concepts currently in AADL. The main advantages of using AADL are the following :

- It makes it possible to apply system engineering approach to software intensive systems rather than brute force.
- The resulting architecture is analyzable and this decreases rework, upgrade coasts as well as program risk and complexity.
- It enables rapid system evolution for complex, real-time, safety critical systems with predictable change to both hardware and software.
- It is a standard and is mature (12 years of DARPA investment and additional experiments) in comparison to other ADL.
- It is extendable : it offers a good foundation for additional capabilities in analysis, automated system integration, systems of systems, distribution, and dynamics.

This article describes an Ada set of packages which allows designers to perform resource requirements analysis of AADL specifications. Our Ada packages are based on Ocarina, an AADL parser distributed by the National Telecommunications Engineering School of Paris (ENST)[26].

First, we propose new AADL properties for the modelling of information frequently used in state of the art of the real time scheduling theory which are missing in the current AADL standard. We can then apply both classical scheduling simulation methods and classical scheduling feasibility tests on AADL specifications in the case of distributed systems with dependent tasks[4, 2, 24] and in the case of POSIX 1003.1b schedulers[8].

[1]See http://www.axlog.fr/R_d/aadl/activites_en.html

Second, we propose several methods to evaluate the memory requirements of a real time embedded system described with AADL. These methods are based on queueing system theory[13]. This article focuses on buffer requirements analysis when AADL specifications contain *event* or *event data* ports.

This article is organized as follows. The AADL language is first described in section 2. An overview of the existing AADL tools is then given in section 3. In section 4, we describe how to perform scheduling analysis on AADL specifications with Cheddar. Section 5 is devoted to memory requirements analysis. Finally, we conclude and give future works in section 6.

2. THE AADL DESCRIPTION LANGUAGE

An AADL specification describes both the hardware part and the software part of an embedded real time system. Basically, an AADL specification is mainly composed of components such as data, threads, processes (the software side of a specification), processors, devices and busses (the hardware side of a specification).

A data component may represent a data structure in the program source text. It may contain sub-programs such as functions or procedures. In this case, an AADL data component can be implemented by an Ada tagged record.

A thread is a sequential flow of control that executes a program. An AADL thread can 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 an address space. In the most simple case, a process owns threads and datas.

Finally, processors, busses and devices represent hardware components hosting one or several applications.

Figure 1 shows a simple example of an AADL specification. This specification contains a shared resource (called $R1$) accessed by two threads (threads $TH1$ and $TH2$). The threads and the shared resource are defined into one address space (process $proc0$). The threads are scheduled by a processor called $cpu0$.

AADL is a typed language. In the case of the Figure 1, *task_type* and *shared_resource_type* are types defined by the designer. One can declare properties which are common for several components. (e.g. *task_type* defines several properties for the threads $TH1$ and $TH2$). A property is defined by a name, a value and a type. Information provided by component properties can be related to the component behavior, its state, the way it will be implemented in Ada or anything else that make it possible to perform AADL specification analysis. The designer can provide properties with most of AADL components.

Figure 1 includes some properties examples :

- The *Source_Text* property indicates that the Ada source code of the threads $TH1$ and $TH2$ can be found in the "task_bodies.adb" file. This kind of properties allows CASE tools to merge the Ada source code given by the user with the one generated from the AADL design.

```
data shared_resource_type
end shared_resource_type;

data implementation shared_resource_type.Impl
    properties
        Concurrency_Control_Protocol =>
            PRIORITY_CEILING_PROTOCOL;
end shared_resource_type.Impl;

thread task_type
    features
        can_access : requires data access shared_resource_type;
end task_type;
thread implementation task_type.Impl
    properties
        Source_Text => "task_bodies.adb";
        Source_Stack_Size => 4092;
        Dispatch_Protocol => Periodic;
        Period => 50;
        Compute_Execution_time => 3 ms .. 3 ms;
        Cheddar_Properties::Dispatch_Absolute_Time => 7;
        Cheddar_Properties::POSIX_
            Scheduling_Policy => SCHED_FIFO;
        Cheddar_Properties::Fixed_Priority => 5;
        Cheddar_Properties::Bound_On_
            Shared_Resource_Blocking_Time => 3;
        Cheddar_Properties::Dispatch_Jitter => 10;
end task_type.Impl;
processor a_cpu
end a_cpu;
processor implementation a_cpu.Impl
    properties
        Scheduling_Protocol => RATE_MONOTONIC;
        Cheddar_Properties::Scheduler_Quantum => 1;
        Cheddar_Properties::Preemptive_Scheduler => true;
end a_cpu.Impl;
process a_proc
end a_proc;
process implementation a_proc.Impl
    subcomponents
        TH1 : thread task_type.Impl;
        TH2 : thread task_type.Impl;
        R1 : data shared_resource_type.Impl;
    connections
        data access R1 -> TH1.can_access;
        data access R1 -> TH2.can_access;
end a_proc.Impl;

system a_system
end a_system;
system implementation a_system.Impl
    subcomponents
        cpu0 : processor a_cpu.Impl;
        proc0 : process a_proc.Impl;
    properties
        Actual_Processor_Binding =>
            reference cpu0 applies to proc0;
end a_system.Impl;
```

Figure 1: A simple AADL specification

- The *Source_Stack_Size* property stores the amount of stack required by the Ada task implementing the threads $TH1/TH2$. The *Scheduling_Protocol* defines the way the processor will be shared between the threads of the application. These two last properties are related to the application behavior at execution time. They may be used by CASE tools which are designed to check before execution time that the system will meet its requirements. Section 4 shows how to perform this kind of test on scheduling requirements.

3. OVERVIEW OF EXISTING AADL TOOLS

AADL has been used in important software tools and its use is planned in several projects.

The Software Engineering Institute has developed an Open Source AADL Tool Environment (OSATE) as a set of plug-ins on top of the open source Eclipse platform[19]. The Eclipse Platform is designed for building integrated development environments (IDEs) that can be used to create applications as diverse as web sites, embedded JavaTM programs, C++ programs, and Enterprise JavaBeansTM. The initial set of plug-ins provides a toolset for front-end processing of AADL models :

- A syntax-sensitive text editor with syntax highlighting, popup help.
- A parser (based on open source ANTLR. See www.antlr.org) and semantic checker for textual AADL with conversion into AADL XML and error reporting integrated with the text editor.
- An AADL XML viewer and editor.
- A syntax-sensitive structural object editor of AADL models with drag-and-drop as well as undo capabilities, and an AADL properties viewer; this editor supports both an AADL library view and a system instance view.
- An AADL XML to textual AADL converter (AADL unparser). Additional plug-ins to extend OSATE, including AADL to MetaH conversion, an example analysis plug-in performing security-level checks, and a plug-in performing various consistency checks.
- A graphical AADL editor based on the Eclipse Graphical Editing Framework (GEF. See www.eclipse.org/gef) is being developed by Mayur Patel at USC as an additional front-end plug-in.

TNI Europe supplies State of the Art software modelling tools (UML 2.0/OOA/OOD/Object Oriented) for the development of mission critical software. These tools offer additional capabilities such as AADL. STOOD is one of the earliest software tools to support AADL[6].

The ASSERT project also involves the use of AADL. ASSERT is an European approach for Proof-Based System Engineering (PBSE). In the context of this project, Axlog develops AADL tools based on OSATE. In the same project, a work aiming at defining and integrating the notion of connector as fundamental element to support dependability uses AADL as the main basis. The latter is done in the distribution and hard real time cluster of the ASSERT project.

To perform AADL analysis, Cheddar relies on Ocarina [26]. Ocarina is a lightweight Ada95 library developed at the National Telecommunications Engineering School of Paris (ENST)[2]. It provides facilities to parse and print AADL files; it also provides an API to navigate through AADL models and instantiate AADL descriptions. Ocarina was created as a fundation library to perform code generation, configuration and deployment for distributed applications described in AADL, in connection with the ASSERT project.

Finally, Axlog has developed ADeS, a software tool to simulate the behavior of an architecture described with AADL. ADeS has been developed in a joint study with the European Space Agency (ESA) to evaluate the interest of AADL for the space domain.

Despite all these projects, there is currently few open source tools devoted to performance analysis of AADL specifications. The next sections describe such tool.

4. SCHEDULING REQUIREMENTS ANALYSIS ON AADL SPECIFICATIONS

In the real time scheduling theory[5], a real time application is defined by a set of processors, shared resources and tasks. In the most simple task model, each task periodically performs a treatment. This "periodic" task is defined by three parameters : its deadline (D_i), its period (P_i) and its capacity (C_i). P_i is a fixed delay between two successive wake up times of task i. Each time the task i is woken up, it has to do a job whose execution time is bounded by C_i units of time. This job has to be ended before D_i units of time after the task wake up time.

From a set of tasks, two kinds of analysis can be performed by Cheddar : scheduling simulation and feasibility tests.

Scheduling simulation consists in predicting for each unit of time, the task to which the processor should be allocated. Checking if tasks meet their deadline can then be done by analyzing the computed scheduling. Figure 2 shows a set of 3 periodic tasks (T1, T2 and T3) respectively defined by the periods 29, 5 and 10, the capacities 7, 1, 2 and the deadlines 29, 5 and 10. These tasks are scheduled with a preemptive Rate Monotonic scheduler[3].

For a given task set, if a scheduling simulation is very long to compute [15], feasibility tests can be applied instead. Different kinds of feasibility tests exist. The most simple test is based on processor utilization factor. With a set of periodic tasks, the processor utilization factor can be computed with the formula $\sum_{i=1}^{n} \frac{C_i}{P_i}$, where n is the number of tasks on the processor. For instance, with a preemptive Rate Monotonic scheduler, Liu and Layland have shown that if the processor utilization factor is less than $n(2^{1/n} - 1)$, task temporal constraints are met [16].

Most of the real time scheduling theory abstractions already exist in the current AADL standard. An AADL thread can be directly implemented by a task such as the one described above and AADL properties can express temporal properties such as deadline, period or capacity (see [9], chapter 5). In the same way, real time scheduling theory shared resources can be implemented by AADL data components

[2]Ocarina is free software, available at `http://ocarina.enst.fr`

[3]With such a scheduler, the task with the lowest period is the task with the highest priority.

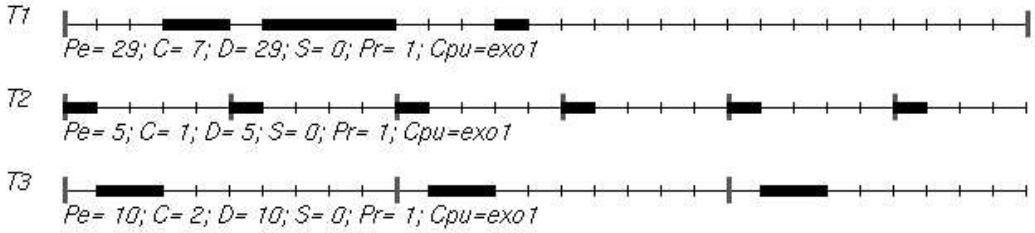

Figure 2: A set of periodic tasks scheduled according to Rate Monotonic

and classical synchronization protocols[20] can be expressed with AADL properties.

Then, most of Cheddar's scheduling analysis services can be directly used. However, some AADL properties were required to run some real time scheduling analysis algorithms. Section 9 gives a sample of AADL properties we implemented into Cheddar. All these new properties were grouped into a set called *Cheddar_Properties* (see Figure 1). The *Cheddar_Properties* set includes properties related to task dependencies, POSIX 1003.1b scheduling and shared resources.

Properties related to tasks dependencies (*Dispatch_Jitter* and *Task_Dependencies*) permit to compute thread response time in the case of distributed systems [24] and to schedule AADL threads according to precedency relationships[4, 2].

POSIX 1003.1b scheduling is based on fixed priority[8]. A POSIX scheduler allocates the processor to the task with the highest priority. If two POSIX threads have the same priority level, the scheduler chooses the task to run according to the task policy and the scheduler quantum. A quantum is a bound on the time a task can hold the processor. Task policy may sort tasks according to the time they become ready to run (ie. $SCHED\_FIFO$). The new AADL properties *Fixed_Priority* and *POSIX_Scheduling_Policy* store for each thread its fixed priority and its scheduling policy. The properties *Scheduler_Quantum* and *Preemptive_Scheduler* are associated to processor components. The first one stores the maximum time a POSIX thread can keep the processor. The second one indicates if the POSIX scheduler works preemptively or not.

Finally, even if some standard AADL properties already exist for shared resources, the current Cheddar's Ada packages introduce new properties such as *Bound_On_Shared_Resource_Blocking_Time* and *Critical_Section* in order to ease the implementation of shared resources analysis algorithms.

```
thread Producer
    features
        Data_Source : out event data port;
end Producer;
thread implementation Producer.Impl
    properties
        Dispatch_Protocol => Periodic;
        Period => 20 ms;
end Producer.Impl;

thread Consumer
    features
        Data_Sink : in event data port;
end Consumer;
thread implementation Consumer.Impl
    properties
        Dispatch_Protocol => Periodic;
        Period => 10 ms;
end Consumer.Impl;

process With_Buffer
end With_Buffer;

process implementation With_Buffer.Impl
    subcomponents
        Producer1 : thread Producer.Impl;
        Producer2 : thread Producer.Impl;
        Consumer1 : thread Consumer.Impl;
    connections
        event data port Producer1.Data_Source -> 
            Consumer1.Data_Sink;
        event data port Producer2.Data_Source -> 
            Consumer1.Data_Sink;
end With_Buffer.Impl;
```

Figure 3: AADL specification with buffers

5. MEMORY REQUIREMENTS ANALYSIS ON AADL SPECIFICATIONS

The previous section described how scheduling analysis can be performed on AADL specifications. In Figure 1, we have seen that the threads of an AADL specification can communicate with shared data. In the AADL standard, thread synchronization and communication can also be expressed with the abstraction of port. Ports are logical connection points between components that can be used for transfer of control and data between threads.

Three kinds of ports exist : *event ports*, *data ports* and *event data ports*. Figure 3 shows an example of threads connected by *event data port*. Even data port are intended for message transmission.

In this example, three threads exchange messages : a thread (called *Consumer*1) receives messages from the two other threads (called *Producer*1 and *Producer*2). All threads are periodic and we can assume that consumption and production rates are periodic too. Depending on the rate threads produce or consume messages, this specification implies that the system will require memory to store in a buffer the set of received messages before their consumption.

A good performance analysis tool should provide features for both scheduling and buffer analysis. Indeed, maximum buffer size must be chosen according to the rate threads

Queue	L	W	P_n
M/M/1	$\frac{\lambda W_s}{1-\rho}$	$\frac{W_s}{1-\rho}$	$(1-\rho)\rho^n$
M/G/1	$\lambda W_s + \frac{\lambda^2(W_s^2+\sigma_s^2)}{2(1-\rho)}$	$\frac{\lambda(W_s^2+\sigma_s^2)}{2(1-\rho)}$	
M/D/1	$\lambda W_s + \frac{\lambda^2 W_s^2}{2(1-\rho)}$	$W_s + \frac{\lambda W_s^2}{2(1-\rho)}$	

Table 1: Main performance criteria

produce or consume data. In the same way, thread consumption or production rate have to be chosen according to the maximum size of the buffer when they will write/read data.

Cheddar provides such buffer analysis tools. They are based on queueing system theory. In the sequel, we propose some buffer analysis tools in order to predict buffer requirements in the case of AADL threads connected by event data ports. The AADL consumer threads are periodic. The AADL producer threads may be aperiodic or periodic. Consumers and producers are both scheduled with real time schedulers such as Rate Monotonic.

5.1 Queueing system theory

The queueing system theory makes it possible to study performance of a system composed of servers, customers and storage places [11] : people waiting in a room for a doctor, network switch routing data, ... If new customers arrive in the system when a server is busy, their requests are stored in a queue. By defining the average rate of customer arrivals and the average rate of requests that the server can handle, the queueing system theory allows the designer to compute many performance criteria. Table 1 gives the main performance criteria for the most usual queueing systems. In this table, λ means the customer arrivals rate, ρ means the queueing system utilization factor, W_s and σ_s^2 respectivelly mean the average customer service time and its variance. With these parameters, one can compute the average number of customers (L), the average customer waiting time (W), and the probability of having n customers in the queue (P_n).

Different customers inter-arrival time distributions and service time distributions exist. The most usuals are deterministic (D), markovian (M) and general (G). D means constant delay between two customer arrivals and constant customer service times. M describes a customer arrival rate or a service time where delays follow an exponential probability distribution. Finally, if no assumption on the probability distribution is done, G is used. G is only defined by an average rate and its variance.

Following the Kendall notation, a queueing system is described by at least 3 parameters : $a|b|c$. The a parameter is the customer arrival rate. b describes the service time rate. Finally, c is the number of servers. For instance, a system with one server, with a constant service time and an exponential client arrival is an M/D/1 queueing system.

Let's go back to the buffer requirements analysis of AADL threads and event data ports. Even if buffers are common operating system functionalities, it seems that few buffer performance analysis results exist that we can apply to AADL periodic threads scheduled according to a real time scheduler [25].

However, some results for similar systems exist. In priority queueing, a priority can be given to customers [1, 23]. The most common priority queue is the HOL (Head Of Line) queue where priorities are fixed [10].

Chen proposes mean waiting time for real time traffic with deadline constraints [3]. This work is based on non preemptive M/G/1 and work-conserving queue. Each real time traffic is a customer with a priority given by the Earliest Deadline First policy.

The real time queueing theory aims at using priority queueing in order to check temporal constraints of tasks randomly activated under "heavy traffic" [14].

None of these approaches suits to AADL periodic threads. Indeed, these approaches assign priorities to customers/messages. Futhermore, they can not handle the fact that thread can be woken up even if no message is stored in buffers, which may be the case of periodic or sporadic AADL threads.

A periodic server can be found in the queueing system theory. Such a queueing system is composed of some queues cyclically served by a single server [21]. Except the periodic behavior of the server, the service time distribution does not handle the fact that thread reponse time can be variable due to the real time scheduler.

Several works on queueing system have been led in the real time community. A lot of queue service disciplines have been studied in the communication field [27]. These services generally aim at providing bandwidth, end-to-end determinist or statistic guarantee on delays. Unfortunately, to avoid buffer overflow, service policies usually proposed in this context have a behavior which depends on the number of messages in the buffer.

5.2 Memory requirements with AADL threads connected by event data port

To study event data port memory requirements when connected AADL threads are scheduled according to a real time scheduler like Rate Monotonic, we propose a new service time distribution : the P distribution. Port buffers are modeled using queueing systems. Customers are messages stored into buffers. The buffer state is modeled by the queueing system server and the queue state. **The P distribution models the fact that periodic consumer/producer threads are scheduled with a real time scheduler**. The P distribution assumes that thread wake up times are not synchronized with message arrivals : **the AADL consumer thread is periodically woken up even if no message is arrived in the buffer.**

```
package Queueing_System is
   ...
   type Queueing_Systems_Type is
      (Qs_Mm1, Qs_Md1, Qs_Mp1, Qs_Mg1, Qs_Pp1,
      ...);
   type Generic_Queueing_System is abstract
       new Ada.Finalization.Controlled with
      record
         Queueing_System_Type : Queueing_Systems_Type;
         Arrival_Rate : Arrival_Rate_Table;
         Service_Rate : Service_Rate_Table;
      end record;
   type Generic_Queueing_System_Ptr is
      access all Generic_Queueing_System'Class;
   ...
   package A_Arrival_Rate is new Indexed_Tables(
      Double, ...);
   package A_Service_Rate is new Indexed_Tables(
      Double, ...);

   subtype Arrival_Rate_Table is
      A_Arrival_Rate.Indexed_Table;
   subtype Arrival_Rate_Range is
      A_Arrival_Rate.Indexed_Table_Range;

   subtype Service_Rate_Table is
      A_Service_Rate.Indexed_Table;
   subtype Service_Rate_Range is
      A_Service_Rate.Indexed_Table_Range;
   ...
end Queueing_System;



package Queueing_System.Theoretical is
   type Generic_Queueing_System_Theoretical is abstract
      new Generic_Queueing_System with null record;
   - Return the theoretical average/maximum waiting time
   - of one customer in the queue
   -
   procedure Qs_Maximum_Waiting_Time (
      A_Queue  : in Generic_Queueing_System_Theoretical;
    Value : in out Double) is abstract;
   procedure Qs_Average_Waiting_Time (
      A_Queue  : in Generic_Queueing_System_Theoretical;
    Value : in out Double) is abstract;

   - Return the theoretical average/maximum number
   - of customer in the queue
   -
   procedure Qs_Maximum_Number_Customer (
      A_Queue  : in Generic_Queueing_System_Theoretical;
    Value : in out Double) is abstract;
   procedure Qs_Average_Number_Customer (
      A_Queue  : in Generic_Queueing_System_Theoretical;
    Value : in out Double) is abstract;

   ...
end Queueing_System.Theoretical;
```

Figure 4: Queueing_System package specifications

```
package Queueing_System.Theoretical.Pp1 is

   type Pp1_Queueing_System_Theoretical is new
      Generic_Queueing_System_Theoretical
     with null record;
   type Pp1_Queueing_System_Theoretical_Ptr is
      access all Pp1_Queueing_System_Theoretical'Class;

   procedure Qs_Maximum_Waiting_Time (
      A_Queue : in out Pp1_Queueing_System_Theoretical;
    Value : in out Double);
      ...
end Queueing_System.Theoretical.Pp1;



package body Queueing_System.Theoretical.Pp1 is
   ...
   procedure Qs_Maximum_Number_Customer (
      A_Queue : in Pp1_Queueing_System_Theoretical;
    Value : in out Double) is
   begin
      If (harmonic = true) then
         Value:= 2*Service_Rate.Nb_Entries;
      Else
         Value:= 2*Service_Rate.Nb_Entries+1.0;
      End if;
   end Qs_Maximum_Number_Customer;

   procedure Qs_Maximum_Waiting_Time (
      A_Queue : in Pp1_Queueing_System_Theoretical;
    Value : in out Double) is
   begin
      If (harmonic = true) then
         Value:= 2 *
            (1.0/Service_Rate.Entries(0));
      Else
         Value:= 2 *
            (1.0/Service_Rate.Entries(0)) +1.0;
      End if;
   end Qs_Maximum_Waiting_Time;
   ...
end Queueing_System.Theoretical.Pp1;
```

Figure 5: P/P/1 queueing system package example

From the P distribution, two new queueing systems are defined : P/P/1 and M/P/1. An exact resolution of P/P/1 is given and we provide an approximation of the M/P/1[13]. This approximation is based on a M/G/1 queueing model. **AADL Applications sharing buffers can then be studied by both worst case and average case analysis. Worst case analysis can be performed if assumptions are made on message arrival rate. In this case, the system is checked with P/P/1 assuming that a smallest period of message arrivals rate exists. Otherwise, if no worst case assumption is made, we show that average analysis can be realized with M/P/1.**

5.2.1 Average buffer performance analysis : producer threads are aperiodic

Let's suppose now that messages arrive in the event data port buffer at a random rate. **This case occurs when AADL producer threads are aperiodic**.

In the sequel, according to the Kendall notation, a buffer receiving random rate messages and shared by an AADL periodic consumer thread will be modeled with a M/P/1 queueing system [11, 18]. Messages are served in a FIFO manner : the earlier a message arrives, the earlier it is served.

We propose an approximation of the M/P/1 queueing system. This M/P/1 approximation consists in evaluating its average service time W_s and its variance σ_s^2. With W_s and σ_s^2, a M/P/1 queueing system can be modeled with a M/G/1 queueing system. Then, M/P/1 message waiting time and number of messages in the buffer can be computed with the following M/G/1 equations [11] :

$$W = W_s + \frac{\lambda(W_s^2 + \sigma_s^2)}{2(1-\rho)}$$

$$L = \lambda W_s + \frac{\lambda^2(W_s^2 + \sigma_s^2)}{2(1-\rho)}$$

where ρ is the queueing system utilization factor and λ, the message arrival rate.

The M/P/1 mean service time and its variance which are valid for all ρ values are[13] :

THEOREM 1. *The M/P/1 average service time is equal to :*

$$W_s = \frac{P_{cons}}{2}(1+\rho) = \frac{P_{cons}}{2(1-\lambda\frac{P_{cons}}{2})}$$

And the variance of the average service time is :

$$\sigma_s^2 = \rho.\left(\frac{1}{n}\sum_{i=1}^{n} S_i^{'} - W_s^2\right) + (1-\rho).\frac{P_{cons}^2}{12}$$

Where $\rho = \lambda W_s$, $S_i^{'}$ are the service time when ρ tends to 1 and P_{cons} is the period of the consumer thread.

Due to the fact that the server models the periodic behavior of the consumer task, the number of $S_i^{'}$ elements is bounded (n elements).

5.2.2 Worst case buffer analysis: producer threads are periodic

We now study a system where event data port buffer productions and consumptions are assumed to be periodic. **This case occurs when AADL producer threads are periodic**.

According to the Kendall notation, a buffer shared by n periodic producer AADL threads and 1 periodic consumer AADL thread scheduled with a real time scheduler can be modeled with a P/P/1 queueing system. Messages are served in a FIFO manner.

Some similarities exist between this system and voice transmission service provided by the AAL1 layer of ATM networks [7]. In order to solve our P/P/1 queueing system, we apply results from this ATM layer.

For a buffer shared by n periodic producers and one periodic consumer, the buffer bound is given by [12] :

THEOREM 2. *For a P/P/1 buffer shared by an harmonic threads set [4] and $\forall i : D_i \leq P_i$, the maximum buffer size and the maximum memorization delay are respectively :*

$$L_{max} = 2.n$$
$$and$$
$$W_{max} = 2.n.P_{cons}$$

For a non harmonic threads set, the maximum buffer size and the maximum memorization delay are respectively :

$$L_{max} = 2.n + 1$$
$$and$$
$$W_{max} = (2.n+1).P_{cons}$$

5.3 Ada implementation of AADL memory requirements analysis

In order to develop AADL event data port memory requirements analysis tools, we wrote a set of object oriented Ada packages (see Figure 6). These packages implement classical queueing system theory results (see Table 1) and results from our works on M/P/1 and P/P/1 queueing systems (see Theorem 1 and 2). These results (called *theoretical* or *analytical* in the sequel) give to the designer a quick estimation of the memory requirements. When any analytical/theoretical performance criterion was never proposed for a given queueing system, the framework also provides a set of Ada packages in order to design, write and run simulations.

Figures 4 and 5 give samples of this set of packages.

Each queueing system is implemented by a tagged record composed of the following attributes :

- *Queueing_System_Type*. This attribute specifies the type of the queue (ie. M/M/1, M/G/1, M/D/1, P/P/1 or M/P/1).
- *Arrival_Rate* and *Service_Rate* arrays. These attributes are used for the specification of mean arrival rates and mean departure rates of customers. *Arrival_Rate_Table* and *Service_Rate_Table* types define such arrays.

The queueing system class is implemented in the *Queueing_Systems* package and its child packages. The *Queueing_System* packages gather all queueing system common

[4] A thread set is said to be harmonic if and only if each thread period is a positive integer multiple of all smaller thread periods.

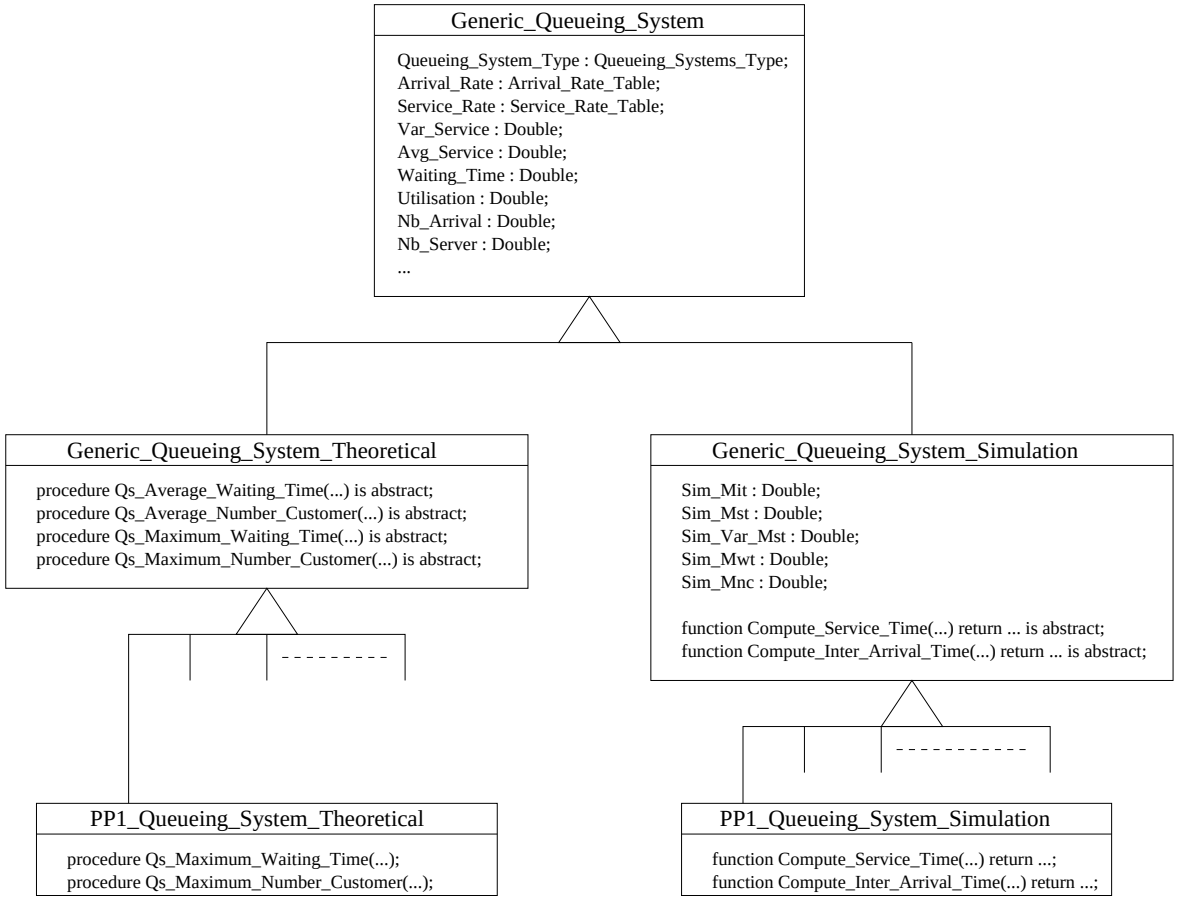

Figure 6: UML diagram of the queueing system framework

source code. These packages contain two inheritance subtrees : one for theoretical/analytical criterion ($Generic\_Queueing\_System\_Theoretical$) and one for simulation ($Generic\_Queueing\_System\_Simulation$).

Most of the time, developing a new queueing system consists in extending this class to add new theoretical/analytical criteria or simulation code. For example, in the case of the theoretical/analytical queueing system inheritance sub-tree, $Qs\_Average\_Waiting\_Time$, $Qs\_Maximum\_Waiting\_Time$, $Qs\_Average\_Number\_Customer$ and $Qs\_Maximum\_Number\_Customer$ abstract sub-programs must be implemented in order to compute customer average waiting time, maximum waiting time, average number of customers and maximum number of customers for each queueing system. For some queueing systems, other theoretical/analytical performance criteria can be computed like probability to be in a given queue state (method $Get\_Probability\_Of\_State$) or probability of a queue overflow (method $Get\_Probability\_Of\_Full\_Buffer$).

If no theoretical/analytical criterion exists for a given queueing system, simulated performance criterion can be computed with the simulation queueing system inheritance subtree. These packages implement an event discrete time queueing system simulator which works as follows (see Figure 7) :

- At initialization step (box 0), the first customer arrival and departure dates are calculated.
- At the first step (box 1), the next event to occur (customer arrival or departure) is chosen.
- At the second step (box 2 and 3), whether the next event is an arrival or a departure (see $Test1$), the number of customers present in the queueing system is decreased or increased by one and the event date is poped or pushed in a buffer for waiting time computation.
- At the third step (box 4 and 5), whether an arrival or a departure has been processed (see $Test2$), a new arrival or departure date is computed with $Compute\_Service\_Time$ and $Compute\_InterArrival\_Time$ functions. These abstract functions have to be implemented for each queueing system in order to compute next customer arrival and departure date.
- At the final step (box 6), from the number of customers gathered (during box 2 and 3), the performance criteria are computed.

6. CONCLUSION AND FUTURE WORKS

In this article, we have presented how performance analysis of AADL specifications can be performed with Cheddar. Cheddar provides a set of Ada packages allowing scheduling and memory requirements analysis. First, we have studied how the scheduling analysis services implemented into Cheddar can be applied to AADL specifications. We have

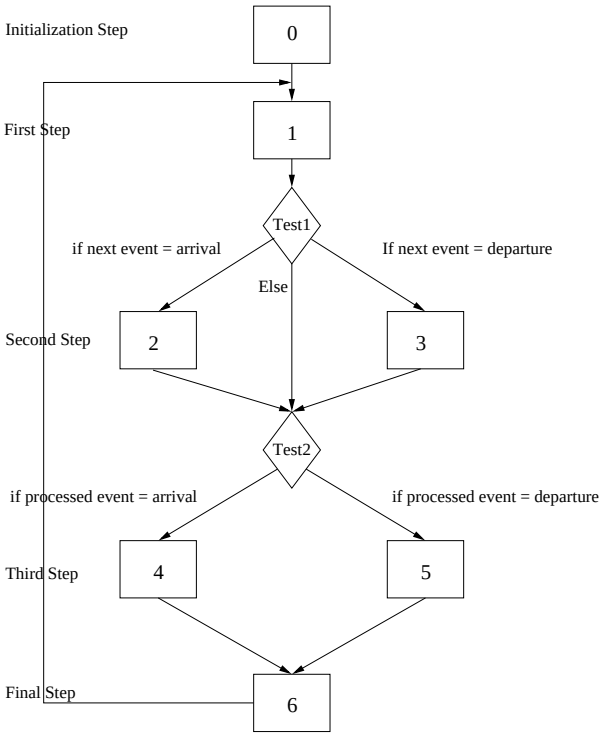

Figure 7: Simulator algorithm Chart

then proposed some new AADL properties to take the state of the art real time scheduling into account. Second, we have studied how memory requirements analysis on AADL specifications can be performed. This article only focuses on memory requirements related to AADL event data ports.

The current Cheddar implementation is able to transform Cheddar's specification into AADL specification. We are currently testing the Ada packages to translate AADL specifications into Cheddar's specification in order to perform scheduling and memory analysis.

These AADL performance analysis methods will be tested with a real life application : a robot monitoring control system called PILOT[17]. This control system is dedicated to the remote control of robot such as AUV (Autonomous Underwater Vehicle). It has been designed and built by the LISYC laboratory in order to ease and to secure the programming of robot missions. We are currently refactoring the PILOT control system software with STOOD[6], an UML/HOOD/AADL models editor distributed by TNI-Europe.

7. ACKNOWLEDGMENTS

We would like to thank the students of the University of Brest who worked on this project and the ENST Ocarina's Team (Thomas Vergnaud, Laurent Pautet and Fabrice Kordon).

8. REFERENCES

[1] K. E. Avrachenkov, N. O. Vilchensky, and G. L. Shevlyaokov. Priority queueing with finite buffer size and randomized push-out mechanism. Technical report, INRIA technical Report number 4434, Mar. 2002.

[2] J. Blazewicz. Scheduling Dependant Tasks with Different Arrival Times to Meet Deadlines. In. Gelende. H. Beilner (eds), Modeling and Performance Evaluation of Computer Systems, Amsterdam, Noth-Holland, 1976.

[3] K. Chen and L. Decreusefond. An Approximate Analysis of Waiting Time in Multi-classes M/G/1/./EDF Queues. 24(1), May 1997.

[4] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic Scheduling of Real-time Tasks Under Precedence Constraints . *Real Time Systems, The International Journal of Time-Critical Computing Systems*, 2(3):181–194, September 1990.

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

[6] P. Dissaux. Using the AADL for mission critical software development. *2nd European Congress ERTS, EMBEDDED REAL TIME SOFTWARE - 21, 22 and 23 January 2004, Toulouse.*

[7] M. Gagnaire and D. Kofman. *Réseaux Haut Débit : réseaux ATM, réseaux locaux, réseaux tout-optiques.* Masson-Inter Editions, Collection IIA, 1996.

[8] B. O. Gallmeister. *POSIX 4 : Programming for the Real World* . O'Reilly and Associates, January 1995.

[9] S. 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, Aug. 2004.

[10] L. Kleinrock. *Queueing Systems : Computer Application.* Wiley-interscience, 1975.

[11] L. Kleinrock. *Queueing Systems : theory.* Wiley-interscience, 1975.

[12] J. Legrand, F. Singhoff, L. Nana, L. Marcé, F. Dupont, and H. Hafidi. About Bounds of Buffers Shared by Periodic Tasks : the IRMA project. In the 15th Euromicro International Conference of Real Time Systems (WIP Session), Porto, July 2003.

[13] J. Legrand, F. Singhoff, L. N. Tchamnda, and L. Marcé. Performance Analysis of Buffers Shared by Independent Periodic Tasks. *LISYC Technical report number legrand-02-2004*, January 2004.

[14] J. P. Lehocsky. Real Time Queueing Theory. pages 186–194. Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96), Washington, DC, USA, December 1996.

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

[16] 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.

[17] L. Nana, F. Singhoff, J. Legrand, J. Vareille, P. L. Parc, F. Monin, D. Massé, L. Marcé, J. Opderbecke, M. Perrier, and V. Rigaud. Embedded intelligent supervision and piloting for oceaographic auv. *IEEE Oceanic Engineering Society - OCEANS'05, France, Brest*, June 2005.

[18] T. G. Robertazzi. *Computer Networks and Systems :*

queueing theory and performance evaluation. Springer-Verlag, 1990.

[19] SEI. OSATE : An extensible Source AADL Tool Environment. *SEI AADL Team technical Report*, December 2004.

[20] L. Sha, R. Rajkumar, and J. Lehoczky. Priority Inheritance Protocols : An Approach to real-time Synchronization. *IEEE Transactions on computers*, 39(9):1175–1185, 1990.

[21] M. Sidi, H. Levy, and S. Fuhrmann. A Queueing Network with a Single Cyclically Roving Server. *Queueing systems, Theory and Applications*, 11:121–144, 1992.

[22] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar : a Flexible Real Time Scheduling Framework. International ACM SIGADA Conference, Atlanta, November 2004.

[23] I. Stavrakakis. A Considerate Priority Queueing System with Guaranteed Policy Fairness. pages 2151–2159. In the proceedings of IEEE Infocom'92 Conference, Florence, May 1992.

[24] K. W. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2-3):117–134, April 1994.

[25] P. Tsigas and Y. Zhang. Non-blocking data sharing in multiprocessor real-time systems. *RTCSA99*, 1999.

[26] T. Vergnaud, L. Pautet, and F. Kordon. Using the AADL to describe distributed applications for middleware to software components. *Ada Europe 2005, 20-24 June, York*, June 2005.

[27] H. Zhang and D. Ferrari. Rate-Controlled Service Disciplines. *In journal of High Speed Netwokrs*, 4(3), 1994.

9. ANNEX : CHEDDAR'S AADL PROPERTIES

```
property set AADL_Project is
   ...
   Supported_Dispatch_Protocols : type enumeration (Periodic, Aperiodic, Sporadic,
      Background, Poisson_Process, Parametric);
   Supported_Concurrency_Control_Protocols : type enumeration (No_Protocol, Priority_Ceiling_Protocol,
      Priority_Inheritance_Protocol);
   Supported_Scheduling_Protocols : type enumeration (Parametric, Earliest_Deadline_First,
      Least_Laxity_First, Rate_Monotonic, Deadline_Monotonic, Highiest_Priority_First);
   ...
end AADL_Project;



property set Cheddar_Properties is

   Dispatch_Seed_is_Predictable : aadlboolean
      applies to (thread, thread group);
   Dispatch_Seed_Value : aadlinteger
      applies to (thread, thread group);
   Dispatch_Absolute_Time : aadlinteger
      applies to (thread, thread group);
   Bound_On_Shared_Resource_Blocking_Time : aadlinteger
      applies to (thread, thread group);
   Dispatch_Jitter : aadlinteger
      applies to (thread, thread group);
   Fixed_Priority : aadlinteger 0..255
      applies to (thread, thread group);
   POSIX_Scheduling_Policy : enumeration (SCHED_FIFO, SCHED_RR, SCHED_OTHERS)
      applies to (thread, thread group);

   Scheduler_Quantum : aadlinteger
      applies to (processor);
   Preemptive_Scheduler : aadlboolean
      applies to (processor);

   Shared_Resource_State : aadlinteger
      applies to (data);

   Critical_Section : list of aadlstring
      applies to (process);
   Source_Global_Heap_Size : aadlinteger
      applies to (process);
   Source_Global_Stack_Size : aadlinteger
      applies to (process);
   Source_Global_Text_Size : aadlinteger
      applies to (process);
   Source_Global_Data_Size : aadlinteger
      applies to (process);

   Task_Dependencies : list of aadlstring
      applies to (system);

end Cheddar_Properties;
```