Implementing an AADL performance analyzer

Frank Singhoff, Laurent Tchamnda Nana, Jérôme Legrand

IMPLEMENTING AN AADL PERFORMANCE ANALYZER

F. Singhoff, L. Nana, J. Legrand

LISYC, EA 3883, University of Brest. 20, avenue le Gorgeu, CS 93837; 29238 BREST Cedex 3, France.
{singhoff,nana,jlegrand}@univ-brest.fr

ABSTRACT

This paper presents a tool we're developing at the University of Brest. This tool is devoted to the performance analysis of AADL specifications. AADL (Architecture Analysis and Design Language) is the AS-5566 standard published by SAE (Society of Automotive Engineers). AADL is a language which makes possible the description of both the hardware and the software parts of an embedded system. From this kind of description, one can generate the software part of the system, but can also perform different kinds of analysis. The work presented in this paper describes how a performance analyzer, called Cheddar, is able to perform such analysis on AADL specifications.

Key words: Performance analysis, AADL, Rate monotonic, Queueing systems.

1. INTRODUCTION

The Architecture Analysis 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 by the Society of Automotive Engineers (SAE) in 2004 [11]. The AADL Standard has been developed by a broad-based international group, including Airbus, the US Army, major avionics companies such as Honeywell, Rockwell Collins, and Smiths Aerospace, internationally recognized experts in UML and Ada, major academic organizations such as the Software Engineering Institute (SEI) and the University of Southern California, and tool suppliers such as Ellidiss Technologies, Artisan Software, High Integrity Solutions and Axlog.

AADL is used to design and analyze the software and hardware architecture of embedded real-time systems and properties that are critical to the operation of such a system such as timing, throughput, and reliability. AADL is applicable to any performance-critical embedded real-time system in domains such as avionics, aerospace, automotive, and autonomous systems.

The main advantages of using AADL are the following:

- It makes it possible to apply system engineering approach to software intensive systems.
- The resulting architecture is analyzable and this decreases rework which upgrade costs 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 (more than 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.

AADL has been used in important software tools and its use is planned in several projects. As example, we can mention OSATE (www.aadl.info), an open source AADL tool environment developed on top of the open source Eclipse platform (www.eclipse.org) by the SEI ; STOOD (http://www.ellidiss.com), a platform developed by Ellidiss Technologies which supplies an integrated support of HOOD, UML 2.0 and AADL with code and documentation generators for the development of mission critical software ; ADeS (http://www.axlog.fr) (Architecture Description Simulation), a software tool developed by Axlog to simulate the behavior of an architecture described with AADL ; TOPCASED (Toolkit in OPen source for Critical Applications and SystEms Development) a project initiated in October 2004 by the CNRT (French National Center of Technological Research) Aeronautic and Space partners and aiming at developing an open source CASE environment (http://www.topcased.org) in order to facilitate the cooperation of tools dedicated to embedded critical

systems and finally, the ASSERT European project which aims at defining and implementing an innovative methodology for the production of embedded real time systems in the aerospace and aeronautics domains for the years 2020. These tools provide features for specification of applications with AADL. Some of them also provide analysis tools but few of them focus on performance analysis.

This paper presents Cheddar, a tool which provides services to do performance analysis. This AADL analyzer is based on Ocarina, an AADL Ada95 parser distributed by the French National School of Telecommunications (ENST Paris). This paper is organized as follows. The usual performance analysis methods implemented in our AADL analyzer are described in section 2. In section 3, examples of AADL analysis with Cheddar are shown. Section 4 is dedicated to conclusions and future works.

2. USUAL PERFORMANCE ANALYSIS METHODS OF EMBEDDED REAL TIME APPLICATIONS

Since 1980, to analyze performance of an application made of concurrent tasks, many models, methods and tools were proposed (eg. Petri Net [9], Synchronous languages [17], ...). In this section we will consider in particular two performance analysis approaches: an approach based on the scheduling theory and an approach based on queueing systems analysis.

2.1. Rate Monotonic Analysis (RMA)

RMA is part of a larger set of quantitative methods: the real time scheduling theory. This theory helps the system designer to predict the timing behavior of a set of real time tasks with scheduling simulation and feasibility tests. Scheduling simulation requires, first to compute a scheduling on a given time interval and second, to look for timing properties in this computed scheduling. On the contrary, feasibility tests allow the designer to study a set of real time tasks without computing scheduling. The first real time scheduling theory contributions were proposed 30 years ago [7]. The theory was strongly extended to cope with many application requirements and was successfully used in many projects [6].

In the classic Liu and Layland real time tasks model [7], each task periodically performs a treatment. This periodic task i is defined by three parameters: its deadline (Di), its period (Pi) and its capacity (Ci). Pi is a fixed delay between two wake up times of the task i. Each time the task i is woken up, it has to do a job whose execution time is bounded by Ci units of time. This job has to be ended before Di units of time after the task wake up time.

From a set of tasks, two kinds of analysis can be performed: 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 be done by analyzing the computed scheduling. Different kinds of feasibility tests exist: tests based on processor utilization factor [7] and tests based on task response time which are designed to check task deadlines [2,3,8] ; tests based on buffer utilization factor which are designed to check buffer overflow [5,10].

An example of feasibility test consists in comparing the worst case response time of each task with its deadline. In [3,8], Joseph, Pandia, Audsley et al. have proposed a solution for the computation of the worst case response time. Tindell et al. have shown in [18] how that solution can be extended to compute worst case response times of task running on distributed systems. Finally, Legrand, Singhoff, et al. have proposed solutions which allow to check buffer overflow for applications made of tasks sharing buffers [5,10]. These solutions also make use of the worst case response time.

2.2. Queueing systems approach

The queueing system theory allows to study performance of a system composed of servers, customers and storage places [12]: people waiting in a room for a doctor, network switch routing data, ...

If customers arrive in the system when a server is busy, their requests are stored in a queue. By defining the average rate of customers request arrivals and the average rate of requests that the server can handle, a queueing system model allows to predict, the average system occupation factor L, the average customer waiting time W, and the probability Pn of having n customers in the queue.

Several works on queueing systems have been done in the real time community. In priority queueing [13], a priority can be given to customers. The most common priority queue is the HOL where priorities are fixed. The real time queueing theory (RTQT [14]) aims at using priority queueing in order to check temporal constraints of tasks randomly activated under heavy traffic (a queueing system with a high utilization factor). A lot of queue service disciplines have been studied in the network field [15]. These services generally aim at providing bandwidth, end-to-end delay determinist or statistic guarantee. Unfortunately, they are based on software or hardware mechanisms which are specific to network switch and consequently difficult to reuse for other application domains.

```
thread implementation t1
    properties
      Dispatch_Protocol => Periodic;
      Compute_Execution_Time => 1 ms .. 2 ms;
      Deadline => 10 ms;
      Period => 10 ms;
end t1;
thread implementation fifo1
    properties
      Dispatch_Protocol => Background;
      Compute_Execution_Time => 1 ms .. 3 ms;
      Cheddar_Properties::POSIX_Scheduling_Policy
        => SCHED_FIFO;
      Cheddar_Properties::Fixed_Priority => 5;
      Cheddar_Properties::Dispatch_Absolute_Time
        => 4 ms;
      Deadline => 100 ms;
end fifo1;
process implementation proc0
    subcomponents
      t1 : thread t1;
....
processor implementation rma_cpu
    properties
      Scheduling_Protocol
        => Rate_Monotonic_Protocol;
      Cheddar_Properties::Preemptive_Scheduler
        => true;
      Cheddar_Properties::Scheduler_Quantum
        => 0 ms;
end rma_cpu;
processor implementation posix_cpu
    properties
      Scheduling_Protocol
          => POSIX_1003_HPF_Protocol;
      Cheddar_Properties::Preemptive_Scheduler
        => true;
      Cheddar_Properties::Scheduler_Quantum
        => 2 ms;
end posix_cpu;
```

Figure 1. AADL thread scheduling analysis

```
data implementation shaded.i
    properties
      Cheddar_Properties::Data_Concurrency_State
        => 1;
      Concurrency_Control_Protocol
        => PRIORITY_CEILING_PROTOCOL;
end shaded.i;
data implementation black.i
    properties
      Cheddar_Properties::Data_Concurrency_State
        => 1;
      Concurrency_Control_Protocol
        => PRIORITY_CEILING_PROTOCOL;
end black.i;
thread J1
    features
      shaded_fea : requires data access shaded.i;
end J1;
thread J2
    features
      black_fea : requires data access black.i;
end J2;
thread J4
    features
      shaded_fea : requires data access shaded.i;
      black_fea : requires data access black.i;
end J4;
thread J5
    features
      black_fea : requires data access black.i;
end J5;
process implementation proc0.i
    subcomponents
      J1 : thread J1.i;
      J2 : thread J2.i;
      J3 : thread J3.i;
      J4 : thread J4.i;
      J5 : thread J5.i;
      shaded : data shaded.i;
      black : data black.i;
    connections
      data access shaded -> J1.shaded_fea;
      data access black -> J2.black_fea;
      data access shaded -> J4.shaded_fea;
      data access black -> J4.black_fea;
      data access black -> J5.black_fea;
    properties
      Cheddar_Properties::Critical_Section => (
        "shaded",
        "J1","2","2",
        "shaded",
        "J4","2","5",
        "black",
        "J2","2","2",
        "black",
        "J4","4","5",
        "black",
        "J5","2","5" );
end proc0.i;
```

Figure 2. AADL threads sharing data ; analysis of shared data usage and thread waiting time

```
thread Producer
    features
      Data_Source : out event data port;
end Producer;
thread Consumer
    features
      Data_Sink : in event data port;
end Consumer;
thread implementation Producer.i ...
thread implementation Consumer.i ...
process implementation p0.i
    subcomponents
      Producer1 : thread Producer.i;
      Producer2 : thread Producer.i;
      Consumer1 : thread Consumer.i;
    connections
      event data port Producer1.Data_Source =>
        Consumer1.Data_Sink;
      event data port Producer2.Data_Source =>
        Consumer1.Data_Sink;
end p0.i;
```

Figure 3. Events exchange with AADL event data ports ; analysis of the buffer requirements

3. EXAMPLES OF AADL ANALYSIS

Cheddar implements both Rate Monotonic Analysis and Queueing Systems Analysis. With a model transformation, AADL specifications are transformed into a set of tasks, processors, temporal constraints, customers, servers and queues. Then, Rate Monotonic and Queueing Systems Analysis can be conducted.

An AADL specification may be composed of components such as threads, data, processes or processors. A thread is a flow of control that executes a program. This kind of component may be implemented by a POSIX thread or by an Ada task. An AADL data models any data structure in a program. Such component may be seen as an UML class. A process can be used by the designer to model an address space protection unit. Finally, the processor components model the execution environment of the programs of an AADL model. An AADL specification may also contain component connections and component properties. Component connections model component relationships. Component properties provide information related to the way components will be implemented, related to their resource requirements, related to their behaviour or anything else which is required in order to build and analyze the modeled system.

Let see some AADL examples in order to show what kind of performance analysis Cheddar can perform.

3.1. AADL thread scheduling analysis

The specification of Fig. 1 declares 2 threads: $t1$ and $fifo1$. The first thread is a periodic thread. This periodic thread is defined with the standard AADL properties: the *Dispatch_Protocol* property means that the thread is a periodic one ; the *Deadline*, *Period* and *Compute_Execution_Time* properties respectively define the deadline, the period and the capacity of the thread (see section 2.1). The second thread is an aperiodic POSIX 1003.1b thread and shows some examples of new AADL properties we defined in order to model and analyze such kind of threads. These new properties are *Preemptive_Scheduler*, *Scheduler_Quantum*, *Fixed_Priority*, *POSIX_Scheduling_Policy* and *Dispatch_Absolute_Time*. They make possible to model and analyze a system built with a POSIX 1003.1b scheduler [16].

An example of analysis result is shown in Fig. 4. In the top part of the window, one can see a set of time lines displaying the scheduling computed by the AADL analyzer. From this set of time lines, the analyzer computes worst/best/average task response times, the number of context switches, the number of preemption and can check if some deadlines are missed. Some other results are displayed in the bottom part of the window: these results are produced with feasibility tests (worst case response time test, processor utilization factor test, ...) and can be considered as a kind of proof. Feasibility tests do not require to compute the simulation and then, can be applied when computing simulation becomes a too much long work to do.

3.2. AADL data analysis

Fig. 2 shows a second AADL specification example. This one is composed of a set of threads sharing data components. This new AADL specification declares two data components: *shaded.i* and *black.i*. They are accessed by a set of threads ($J1$, $J2$, $J3$, $J4$ and $J5$). One more time, new AADL properties were defined in order to make it possible to use the performance analysis tools of Cheddar. These properties are:

- *Data_Concurrency_State*: this property gives the initial state of the data. In Cheddar, a data is seen as a Dijkstra semaphore. As for an initial semaphore value, this property indicates the number of non blocking data access the set of thread can do before being blocked.
- *Critical_Section*: this property stores the set of critical sections of the thread/data components of a given process. A critical section is a piece of thread capacity in which the thread will access

a given data. The *Critical_Section* is a list of 4-uplets (a, b, c, d). Each 4-uplet (a, b, c, d) models a critical section where a is the accessed data, b the considered thread, c and d respectively the start time and the end time of the critical section (relatively to the thread capacity). In the example of 2, five critical sections were modeled for the process $proc0.i$.

Some standard AADL properties are also used in this example. The most important one is *Concurrency_Control_Protocol* which describes how the shared data will be accessed [20].

Fig. 5 shows the simulation results a user can expect from this AADL specification. From a simulation, one can compute data blocking time per thread [20]. A thread blocking time is a delay a thread has to wait before accessing a given data. These delays can also be bounded according to the concurrency control protocol without running simulations: it's a kind of feasibility test. Fig. 5 shows such blocking time. In the top part of the windows, one can see time lines associated to data (*black* and *shaded*) which display when the data components are acquired and released. In the bottom part of the window, from the simulation, the best/worst/average thread blocking times are computed. For example, from this simulation, we learn that the thread $J2$ has to wait 5 units of time in order to access to the *black* data.

3.3. AADL event data port analysis

The last AADL specification in Fig. 3 shows a system composed of threads which exchange messages thought event data port. Event data port are communication channels. They can used for asynchronous message transmission between threads. These messages are called events. Events are queued and usually served with a FIFO policy. Queueing systems may be able to predict event data port memory requirement if queueing models take into account AADL thread dispatching (eg. periodic) and AADL thread scheduling (eg. Rate Monotonic) properties. Cheddar provides feasibility tests based on such queueing systems. The AADL example of Fig. 3 contains a process, called $p0.i$, declaring 3 threads: two producers ($Producer1$ and $Producer2$) and one consumer of events ($Consumer1$). Event data port connections express event exchange relationships between the 3 threads. The first event data port connection says that events sended by $Producer1$ will be read by $Consumer1$. In the same way, the second event data port connection says that events sended by $Producer2$ will be read by $Consumer1$.

As for the previous AADL specifications, Fig. 6 shows the simulation results which can be computed by Cheddar from the AADL specification. The top part of the window displays a new set of time lines which shows when events are sended and received. The bottom part of the window displays the results of feasibility tests based on queueing systems: a worst case number of events in the event data port buffer is computed and displayed.

4. CONCLUSION AND FUTURE WORKS

This paper describes a tool which can be used for performance analysis of systems designed with AADL. The tool is freely available and can be downloaded from http://beru.univ-brest.fr/~singhoff/cheddar. It is based on two usual performance analysis methods: Rate Monotonic Analysis and Queueing Systems Analysis. From these methods, an AADL designer can automatically check if the tasks of his system will meet their temporal requirements and if the buffers of his application are large enough.

This AADL analyzer can be run alone but it can be also used with a CASE tool. For example, Cheddar is known to work with STOOD, an UML/HOOD/AADL design tool distributed by Ellidiss Technologies [2].

To perform AADL analysis, Cheddar relies on Ocarina [1]. Ocarina is a lightweight Ada95 library developed at the National Telecommunications Engineering School of Paris (ENST)[1]. 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 foundation library to perform code generation, configuration and deployment for distributed applications described in AADL, in connection with the ASSERT project.

In the next months, we plan to extend Cheddar to allow designers to perform analysis of AADL systems composed of hierarchical schedulers [19]. Some services related to task precedency relationships and end to end task response time in distributed systems will be also implemented.

5. REFERENCES

[1] 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.

[2] 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.

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

[3] M. Joseph and P. Pandia. Finding Response Time in a Real Time System. Computer Journal, 29(5):390–395, 1986.

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

[5] J. Legrand. Contribution à l'ordonnancement des systèmes temps réel comprenant des tampons (in french). Phd thesis, December 2004. University of Brest.

[6] SEI. The Rate Monotonic Analysis. Technical report, In the Software Technology Roadmap. http://www.sei.cmu.edu/str/descriptions/rma_body.html, September 2003.

[7] C.L. Liu and J.W. Lailand. Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment. Journal of the ACM, 20(1):46-61, January 1973.

[8] A.N. Audsley, A. Burns, M. Richardson and K. Tindell. Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal, pages 284-292. 1993.

[9] J.L. Peterson. Petri Net Theory and the modelling of Systems. Prentice Hall. 1981.

[10] F. Singhoff, J. Legrand, L. Nana, and L. Marc. Extending Rate Monotonic Analysis when Tasks Share Buffers. In the DAta Systems in Aerospace conference (DASIA 2004), Nice, July 2004.

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

[12] L. Kleinrock. Queueing Systems : theory. Wiley-Intersciences. 1975.

[13] L. Kleinrock. Queueing Systems : computer Application. Wiley-Intersciences. 1975.

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

[15] H. Zhang and D. Ferrari. Rate-controlled Service Disciplines. In Journal of High Speed Networks. 4(3). 1994.

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

[17] P.L. Guernic, T. Gautier, M.L. Borgne and C.L. Maire. Programming Real Time Applications with Signal. INRIA-RENNES.Technical Report number 1446. 1991.

[18] 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.

[19] 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. December, 2001, pages 3-14.

[20] Sha, R. Rajkumar and J.P. Lehoczky. Priority Inheritance Protocols : An Approach to Real Time Synchronization. IEEE Transactions on computers, 39(9):1175-1185. 1990.

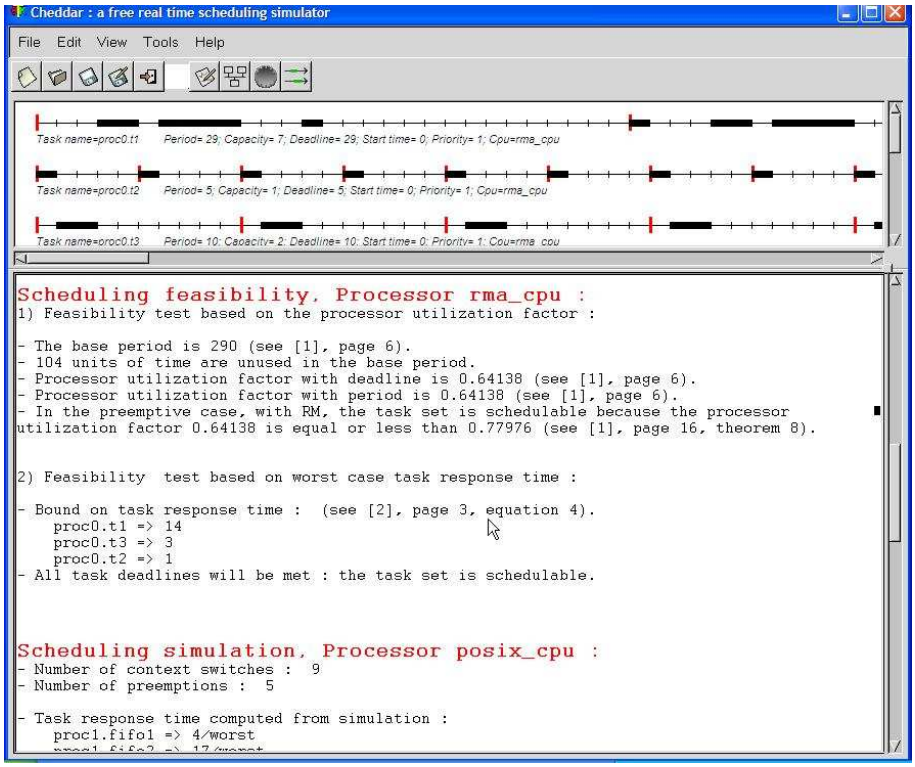

Figure 4. AADL thread analysis

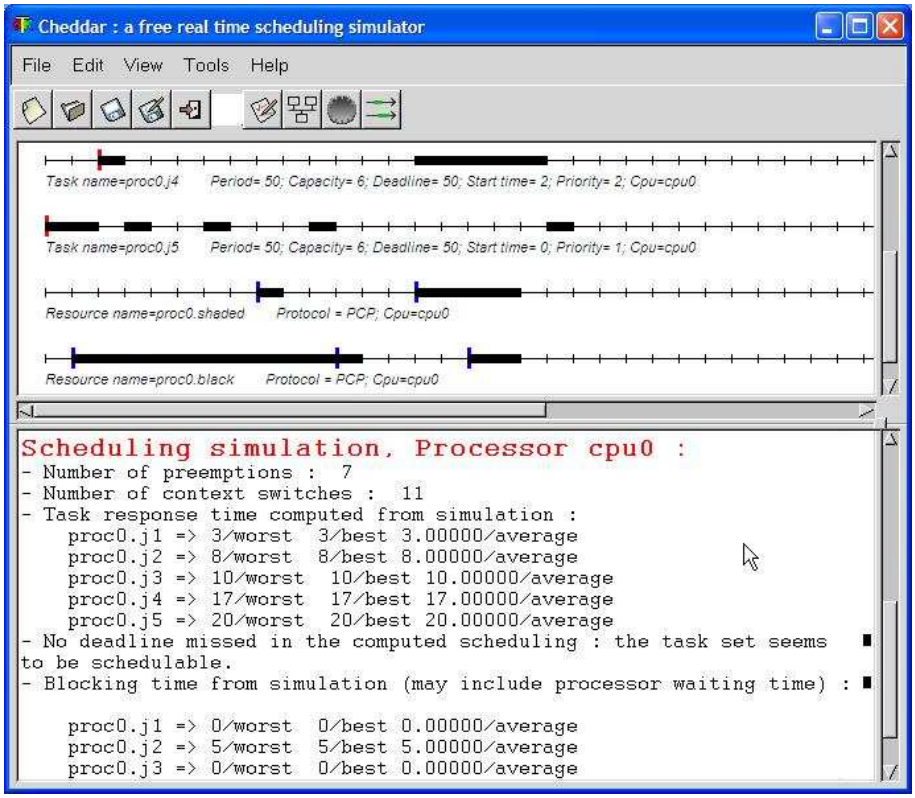

Figure 5. AADL shared data analysis

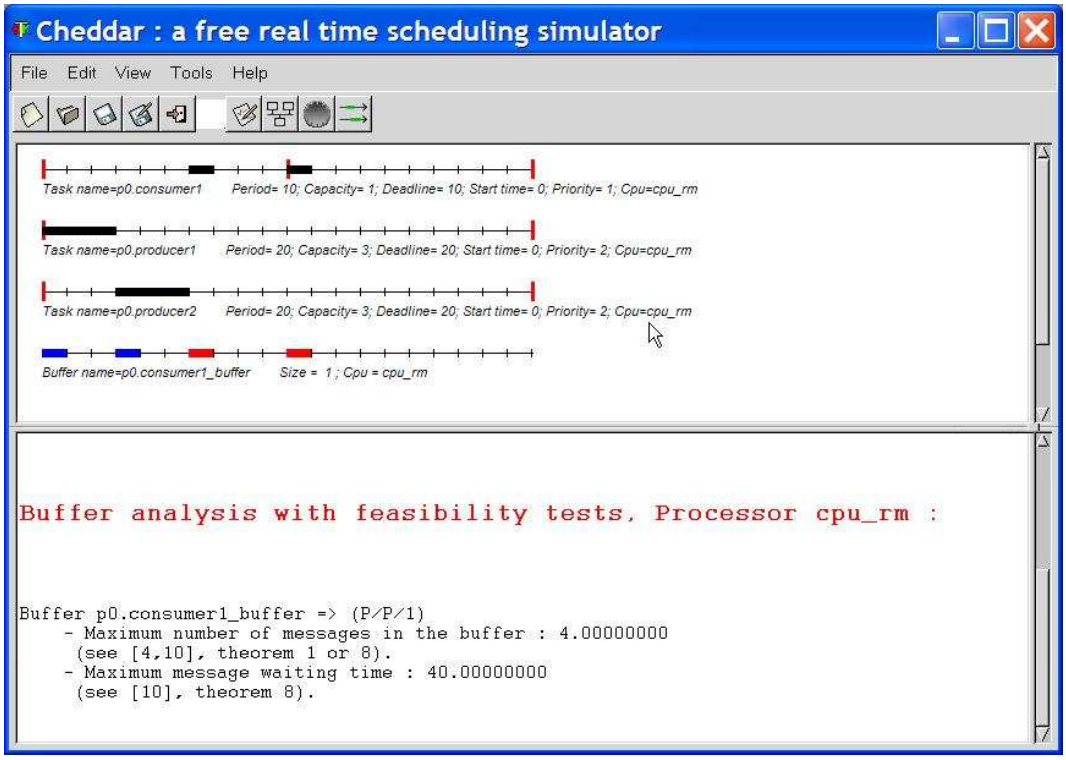

Figure 6. AADL event data port analysis