Ravenscar Computational Model compliant AADL Simulation on LEON2

Roberto VARONA-GÓMEZ, Eugenio VILLAR
TEISA, GIM, Universidad de Cantabria
39005 Santander, Spain
{roberto, villar}@teisa.unican.es

Ana-Isabel RODRÍGUEZ-RODRÍGUEZ
OSV, GMV Aerospace and Defence S.A.U.
28760 Tres Cantos, Spain
airodriguez@gmv.com

ABSTRACT

AADL has been proposed for designing and analyzing SW and HW architectures for real-time mission-critical embedded systems. Although the Behavioral Annex improves its simulation semantics, AADL is a language for analyzing architectures and not for simulating them. AADS-T is an AADL simulation tool that supports the performance analysis of the AADL specification throughout the refinement process from the initial system architecture until the complete, detailed application and execution platform are developed. In this way, AADS-T enables the verification of the initial timing constraints during the complete design process. In this paper we focus on the compatibility of AADS-T with the Ravenscar Computational Model (RCM) as part of the TASTE toolset. Its flexibility enables AADS-T to support different processors. In this work we have focused on performing the simulation on a LEON2 processor.

Keywords: AADL, simulation, Ravenscar, LEON2, SystemC.

1. INTRODUCTION

Architecture Analysis and Design Language (AADL) [1-4] was developed as a Society of Automotive Engineers (SAE) standard to enable the description of task and communication architectures of real-time, embedded, fault-tolerant, secure, safety-critical, SW-intensive systems. It is used to describe the software and hardware components of a system and the interfaces among them.

The Automated proof-based System and Software Engineering for Real-Time systems (ASSERT) project [5] led to a new development process for distributed embedded real-time software, and a set of methods and tools for supporting the process. The process is based on separation of concerns, automatic code generation and property preservation. An important feature of the ASSERT process is the adherence of the concurrency model to the RCM [6], a restricted tasking model that enables static response time analysis of real-time systems. The model restricts the concurrency model to a static set of periodic and sporadic threads communicated by means of a static set of shared data objects, protected by mutual exclusion synchronization mechanisms. There are two variants of the ASSERT software process: Hard Real-Time Unified Modeling Language (HRT-UML) and AADL tracks. The ASSERT Set of Tools for Engineering (TASTE) [7] toolset is an open source toolset supporting the latter.

The LEON2 [8] processor was designed by the European Space Agency (ESA) as a 32-bit synthesizable processor core based on the SPARC V8 architecture. The core is highly configurable, and particularly suitable for System on Chip (SOC) designs.

There is a commonly recognized need for new development frameworks that enable designers to perform efficient exploration of design alternatives and analyze system properties throughout the design cycle. Some system properties can be obtained by static analysis. Many other properties can only be obtained through simulation. In most complex cases, system simulation is necessary for performance analysis under real execution conditions. System simulation validates the correct dimensioning of the system, detection of locks, missed deadlines and other potential problems caused by the complex interaction among components that can be found in a real system. The earlier all those problems are detected, the less is the associated cost of correcting them [9].

Evolutionary prototyping is becoming a well-accepted development approach in Model-Driven Engineering (MDE) [10]. The design flow is based on a central model that is iteratively refined until it is satisfactory. Programs can be generated from this model and constitute intermediate versions of the product. The last model refined corresponds to the final system. A prototyping-based design process is of interest to verify, as early as possible, the impact of deployment decisions, or the use of a particular HW/SW component in the system.

In this paper, an AADL simulation methodology, now compatible with the RCM, is recalled. This methodology has been implemented in the tool AADS-T [11]. AADS is a simulation framework that can support prototype-based

design allowing the functional and non-functional (execution times, power consumption, etc.) verification of the system while it is being refined right through to the final implementation. AADS is based on SystemC, which has become a relevant standard language for modeling and simulation of HW/SW embedded systems [12]. The SystemC framework supports the seamless integration of HW components and an easy optimization of the executive platform.

The contents of the paper are as follows. The following section reviews the state of the art. In Section 3, the previous work carried out with AADS is summarized. Section 4 describes how support is provided to perform the simulation on a LEON2 processor. Then, we explain the main part of this paper, which is the compatibility of AADS-T with the RCM. Next, a case study is presented and finally conclusions are stated.

2. STATE OF THE ART

Several authors have considered ASSERT and the RCM in their research. Some of their papers deal with the ASSERT Virtual Machine (VM), the execution platform on which ASSERT applications run, based on the RCM. J. A. de la Puente et al. [13] and J. Zamorano et al. [14] are good examples of this.

M. Bordin et al. [15] propose some guidelines to generate RCM-compliant Ada code from HRT-UML. S. Mazzini et al. [16] explain a MDE methodology for the development of high-integrity real-time systems. However using UML does not facilitate the low-level description of the system. Besides, the different views of the system use different formalisms, so one must modify all views on each change of the system to get a coherent model, hindering rapid prototyping.

J. Kwon et al. [17] propose Ravenscar-Java, a high-integrity profile for real-time Java. However we think that Java is not a good high integrity programming language due to its object-oriented programming features, its automatic garbage collection, and the proposed limitations to the extension of real-time multi-threading that cause confusion.

Ocarina [10] is a tool suite that uses code generation facilities in Ada and C to analyze AADL models. The code generated is compatible with the RCM.

After analyzing the state of the art, it follows that no approach uses SystemC [18], which is a recognized standard for modeling HW/SW platforms, with its great potential for integration of processors, buses, memories and specific platform HW. Our solution makes HW/SW co-design easier because of the use of SystemC.

SCoPE [19-20] is a C++ library that extends the standard language SystemC without modifying it. It simulates C/C++ SW code based on two different operating system interfaces (POSIX [21-22] and MicroC/OS [23]). Moreover, it co-simulates these pieces of code with HW described in SystemC. SCoPE generates a file with this SystemC description of the model.

In previous works [24-25], preliminary versions of AADS supporting a part of the AADL standard and its Behavioral Annex were developed. Now we have improved AADS to make it compatible with the RCM and to take into account the LEON2 processor. AADS-T supports RCM-compliant AADL simulation in SystemC, thus enabling the HW platform to be modeled and permitting HW/SW co-design. The AADL model is based on Portable Operating System Interface for UNIX (POSIX), so it supports many different Real Time Operating Systems (RTOS).

3. PREVIOUS WORK

This work is an extension of AADS, an AADL simulation tool written in Java, which was developed as a plug-in [26] of Eclipse [27].

AADS enables the modeling of a subset of AADL including the Behavioral Annex for purposes of implementation and simulation. The starting point of the simulator is a functional AADL specification without detailed code. For each component, the corresponding timing constraints are defined. This initial AADL specification supports the verification of the global performance constraints of the system based on the specific timing constraints of the different components. The AADL model is parsed using AADS and a model suitable for simulation with SCoPE is produced, in order to check whether the AADL constraints are fulfilled.

As the design process advances and, on the one hand, the actual functionality is attached to the SW components using the corresponding source code and, on the other, the functionality is mapped onto specific platform resources, a more accurate performance estimation is performed. These refined properties can be added to the AADL model and a new model can be generated by AADS. By comparing the initial timing constraints with these refined, timing estimations, it is possible to verify the non-functional correctness of the design process at any refinement step.

AADL enables the specification of both the architecture and functionality of an embedded real-time system. AADS translates both to SystemC (see Fig. 1). It parses the AADL model including the Behavioural Annex so the functionality is translated to an equivalent POSIX model and the architecture is represented in eXtensible Mark-up Language (XML) [28]. AADS supports multiple HW components (processors, memories, devices and buses).

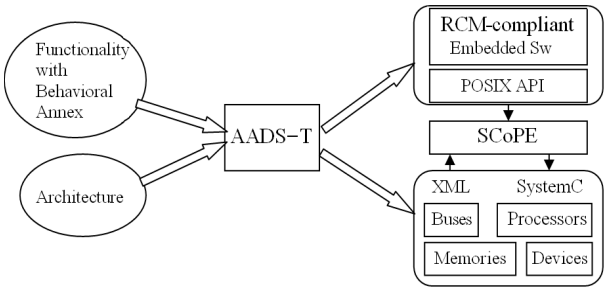

Figure 1. Translation process.

4. LEON2 MODELING

In previous works, SCoPE admitted only some of the Advanced RISC Machines (ARM) family processors. The LEON2 processor was designed by the ESA as a 32-bit synthesizable processor core based on the SPARC V8

architecture. The core is highly configurable, and particularly suitable for SOC designs. SCoPE has been modified to include the LEON2 processor at 50 Mhz, 15.4 Million instructions per second (MIPS) and 30.64 nJ of energy consumed per instruction in its *processors.xml* configuration file. It was necessary to specify the data and instruction cache sizes too. A *size* of 8192, a *size of line* of 8 and an *associativity* of 1, considering an *instruction size* of 4 (32 bits), was considered for both. Another configuration file of SCoPE, *meminst.xml*, was modified to include the operation codes of the LEON2.
The GNU cross-compiler for LEON2 used is the GNAT for the LEON 2.1.0 C compiler so, to be compiled, the source code produced by AADS-T has to comply with certain characteristics of little importance.

5. COMPATIBILITY WITH THE RCM

The real-time behavior specification of ASSERT models is based on the RCM, a model of concurrency for high-integrity systems that enables formal analysis of the temporal properties of a system using response-time analysis techniques. The model includes a static set of concurrent threads of execution, communicating by means of shared protected data with mutually exclusive read and write access, and a restricted form of conditional synchronization. The model is simple enough to be implemented by a simple, small-size real-time kernel, thus easing the way to the eventual certification of real-time systems based on it.
Twelve properties must be fulfilled to be RCM-compliant; the source code generated by AADS-T obeys all of them. These properties are stated in an internal document of the HW-SW CODESIGN project [29] titled R1-4 Evaluation of Compliance with the ASSERT Process, written by Juan Antonio de la Puente and Juan Zamorano.

Basic Elements

There are two main elements in the RCM: threads and protected objects. A thread is the basic unit of execution, which can be executed concurrently with other threads on a single processor. Protected objects are an abstraction of shared data, synchronization, and interrupt handling.
There are a static number of threads and protected objects. Therefore, threads and protected objects can only be created at system initialization time.

RCM 1: A real-time system consists of:

- A static set of N threads, $\mathrm{T} = \{\tau_i\}$, i ∈ 1..N.
- A static set of M protected objects, $\mathrm{O} = \{\theta_i\}$, i ∈ 1..M.

The set O may be empty (M = 0), in which case the system is said to have only independent threads.
In the source code generated by AADS-T all the threads and protected objects are created calling *pthread_create* and as objects of the corresponding classes respectively at system initialization time.

Properties of Threads

A thread is a concurrent unit of execution with the following properties:

RCM 2: Threads are non-terminating. They exhibit an endless repetitive behavior alternating between the following states (see Fig. 2):

- Suspended: a suspended thread is not eligible for execution.
- Ready: a ready task can be executed when the processor is allocated to it.

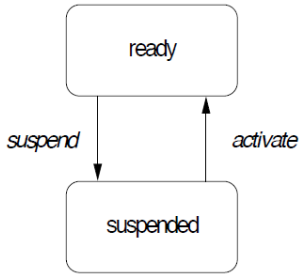

Figure 2. States of RCM threads.

RCM 3: Threads have a single activation point. An activation point is a point in the executable code of a thread at which its state changes from Suspended to Ready.
When activated, a thread becomes ready and then executes a piece of sequential code (thread activity), after which it becomes suspended awaiting the next activation.
Threads of the source code generated by AADS-T use *while(true)* to be non-terminating. They are suspended after executing the sequential code in a *clock_nanosleep* and when sleeping time has passed they become ready at their single activation point.

RCM 4: The activity of a thread is a sequence of code with a bounded and known worst-case execution time (WCET).
The WCET of thread τ_i is C_i.
AADS-T utilizes the AADL property *Compute_Execution_Time* to know the WCET of a thread. The source code generated checks that this WCET is not exceeded.
This property implies that a thread does not execute any operation that could result in its becoming suspended other than the suspension immediately before the activation point. So do the threads created by AADS-T.

RCM 5: A thread can be activated only by one of the following two kinds of events:

- One is by a timing event which is issued periodically by the environment. In this case the thread τ_i is said to be periodic or time-driven with period T_i.
- The other is a synchronization event issued when the barrier of a synchronization protected object is opened (see RCM 8 below). In this case, the thread τ_i is said to be sporadic. The synchronization event must have a minimum inter-arrival time associated to it, i.e. a minimum elapsed time interval between two consecutive occurrences of the event, T_i.

AADS-T uses the AADL properties *Period, Dispatch_Protocol* and *Device_Dispatch_Protocol* to

know the period and the type of a thread respectively. It accepts only *periodic* and *sporadic* threads and not *aperiodic* or *background* threads. The difference between the codes generated is that a sporadic thread waits for an event from an *event* or *eventdata port connection* after invoking a synchronization operation in the activation point. In both cases *clock_nanosleep* waits a time T_i.

Properties of Protected Objects

A protected object is an object which encapsulates a set of data and a set of associated operations (protected operations). The value of the data makes up the state of the object. The state can only be read or changed by invoking one of the operations of the protected object.

If θ is a protected object:

- θ.S denotes its state, θ.S $\in$ **S**, where **S** is an appropriate data domain.
- $\theta . P_k$ denotes the k-th operation of θ.

Notice that a protected object must have at least one operation; otherwise its state is inaccessible.

The notation $\tau \rightarrow \theta$ will be used to denote that τ invokes one or more operations of θ. Similarly, $\tau \rightarrow \theta . P$ means that τ calls the operation $\theta . P$.

AADS-T generates an object of the corresponding class which is a protected object in the source code for each AADL *data, event and eventdata port connection*. Classes generated by AADS-T have the appropriate data members to achieve the communication of data and/or events between threads. Each class has a constructor and member functions read and write to initialize and access data members.

Protected objects have the following properties:

RCM 6: Only one thread can be executing an operation of a given protected object at any given time, i.e. protected operations are mutually exclusive.

Consequently, if a thread invokes a protected operation at a time when another thread is already executing an operation of the same object, it has to wait. When the protected operation that was being executed is completed, the waiting thread is allowed to execute the operation it had invoked. Notice that a thread that is waiting to begin a protected operation is not considered to be suspended. In consequence, a thread activity can invoke protected operations without violating RCM 4.

Each class produced by AADS-T defines a *mutex* that is locked when a member function is called and unlocked when it ends, ensuring compliance with mutual exclusion.

RCM 7: All protected operations have a bounded and known WCET.

The WCET of the protected operation $\theta_i . P_k$ is $C_{i,k}$.

Again, this property implies that no operations that could result in a thread being suspended can be invoked from a protected operation.

AADS-T uses the ad hoc defined AADL properties (newly defined properties in the property set *UC*) *PO_read_WCET* and *PO_write_WCET* for each *port connection* to know the WCET of each member function. The source code generated checks if these WCETs are exceeded. Moreover, no member function calls any suspending operation.

RCM 8: A protected object can have at most one synchronization operation that has an associated barrier, which is a Boolean variable that is part of the object state. When the value of the barrier is true, the barrier is said to be open, and otherwise it is said to be closed.

The behaviour associated with synchronized operations is as follows:

- When a thread invokes a synchronization operation, if the barrier is open the execution proceeds as with an ordinary protected operation; but if the barrier is closed, the thread is blocked.
- At most one thread can be blocked at a barrier at any given time.
- A thread that is blocked at a barrier is unblocked whenever the barrier becomes true (as the result of the execution of another protected operation by some other thread).

Invoking a synchronization operation is a potentially blocking operation, and thus cannot be done within a thread activity; this can only be used to implement the activation events of sporadic threads.

In the source code produced by AADS-T only the objects corresponding to *event* and *eventdata port connections* have a synchronization member function because a sporadic thread is dispatched by an event as stated above. Only sporadic threads invoke the synchronization. The classes corresponding to *event* and *eventdata port connections* have a POSIX condition variable as a datum member that is initialized at system initialization time. The barrier is initialized as false in the constructor, then set to true in the write member function (besides signalling the condition variable to unblock the sporadic thread), then checked to see whether it is false in the synchronization to block the sporadic thread on the condition variable, and finally set to false after unblocking it.

Scheduling

The RCM is associated with an instance of the fixed-priority pre-emptive scheduling (FPPS) method, together with the immediate ceiling priority inheritance protocol (ICPP).

The scheduling model is defined by the following properties:

RCM 9: Each thread τ_i has a basic priority, $P_i \in \mathbf{P} \subset Z$, where Z is the set of the integer numbers.

The basic priority of a thread is fixed, i.e. it is never changed.

AADS-T uses the ad hoc AADL property *Priority* to create a thread at system initialization time with *sched_priority* at that priority, which is never changed.

RCM 10: Each protected object θ_i has a ceiling priority CP_i which is the maximum of the basic priorities of all the threads invoking any of its operations:

$$CP_i = \max P_j, \tau_j \rightarrow \theta_i .$$

As basic priorities of all threads are fixed so too are the ceiling priorities of all protected objects.

RCM 11: At every instant of time, each thread has an active priority. The active priority of a thread is the maximum of:

- The basic priority of the thread, and
- The ceiling priority of all protected objects that contain an operation that is currently being executed by the thread.

Therefore, whenever a thread invokes a protected operation, it immediately inherits the ceiling priority of the enclosing protected object.

In the source code generated by AADS-T the function *pthread_mutexattr_setprotocol* is used with the value *PTHREAD_PRIO_PROTECT* and the function *pthread_mutexattr_setprioceiling* with the maximum of the priorities of the two threads communicating through a *port connection*. This is done when initializing the *mutex* of the object corresponding to that *connection* at system initialization time guaranteeing the fulfillment of RCM 10 and RCM 11.

RCM 12: Ready threads are conceptually grouped into ready queues. There is a ready queue for each priority level in **P**.

Threads are added to and removed from priority queues according to the following rules:

- When a suspended thread becomes ready, it is added at the tail of the priority queue for its active priority.
- When the processor is idle, the thread which is at the head of the non-empty ready queue with the highest priority is dispatched for execution and removed from the queue.
- Whenever there is a non-empty ready queue with a higher priority than the priority of the currently running thread, the thread is pre-empted from the processor and it is added at the head of the ready queue for its active priority.

  Notice that according to the previous rule, the thread at the head of the ready queue that caused the pre-emption is dispatched for execution immediately afterwards.

AADS-T admits only *SCHED_FIFO* for the ad hoc AADL property *POSIX_Scheduling_Policy* of a thread, to set so *sched_policy* in the source code.

The above model specifies a concurrent system with a predictable, analyzable temporal behaviour. Since the execution time of threads is bounded (RCM 4, RCM 7) and the scheduling method is FPPS with ICPP, well-known response-time analysis techniques can be applied to statically guarantee that the system satisfies its temporal requirements.

6. CASE STUDY

AADS-T has been tested in the case study shown in Fig. 3, to assure the feasibility of the translation and its compatibility with RCM. It is a space application of digital image processing that consists of different modules for image processing, science data reporting, control, LEON target on board SW and payload data handling unit. The files produced by AADS-T are compiled with SCoPE to simulate the model and the results obtained are used to refine the model as needed. The simulation executed the source code of the threads and the protected objects enabled the communication among the threads.

During the simulation we have observed that all the threads and protected objects were created at the beginning and their number remained static during the simulation.

We have also observed that the threads do not terminate till the simulation time ends as stated in a parameter for SCoPE. The threads alternate their execution as they become suspended or ready.

The trace of the simulation shows that each thread activity is executed from the corresponding activation point.

We have been able to distinguish clearly between periodic threads and threads activated by an event.

Various protected objects corresponding to different *port connections* (*data, event and eventdata*) between threads have been tested to assure a suitable communication between the threads.

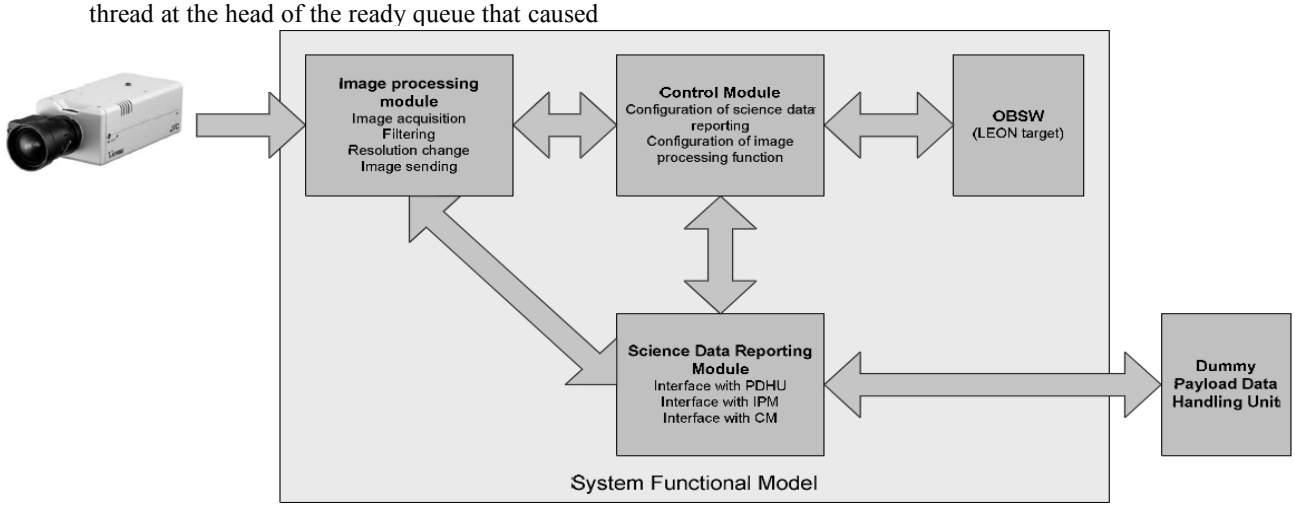

Figure 3. Case Study functional description.

It has been proven that if the WCET of a thread or a protected operation is exceeded, a warning message appears in the trace of the simulation.
Synchronization operations have been tested on sporadic threads dispatched by *event* or *eventdata*.
We have applied different priorities of threads and protected objects to test the scheduling method FPPS with ICPP.
Simulation on a LEON2 processor differs notably after applying AADS+ [25] or AADS-T (see Table I). The AADL model requires more lines to apply AADS-T but there are not as many C++ code lines generated as one might fear. As this code is RCM-compliant, it does not use POSIX *signals* or *message queues* (sending and receiving signals and messages are more time consuming than read and write protected operations), so SCoPE can execute many more instructions and threads and load the bus more. For this reason, the cost in terms of core energy/power, misses, etc is certainly greater.

TABLE I. COMPARISON BETWEEN THE TWO SIMULATIONS' METRICS

	AADS+ not RCM-compliant	AADS-T RCM-compliant
AADL lines	781	846
C++ & XML lines	2801	2962
Number of thread switches	7476	19981
Running time	4922419760 ns	4966811120 ns
Use of CPU	98.4484 %	99.3362 %
Instructions executed	15160730	76508275
Instruction cache misses	5233	91381
Core Energy	4.64524e+08 nJ	2.34422e+09 nJ
Core Power	92.905 mW	468.842 mW
Instruction cache energy	6.99994e+08 nJ	3.57231e+09 nJ
Instruction cache power	139.9988 mW	714.464 mW
Bus access time	836480 ns	14612640 ns
Number of interrupts	7820	22258
Instruction miss transfers	206	1099
Bus Load	168224 bytes	2923240 bytes

7. CONCLUSIONS AND FURTHER WORK

This paper describes the compatibility with RCM of the AADS-T simulation tool. AADS-T supports the refinement of AADL models, including the Behavioral Annex, through performance analysis done with SCoPE, after translating those models.
The generation of the SystemC model from the Ravenscar Computational Model compliant AADL specification is not straightforward. Nevertheless, the SystemC model generated by AADS-T is able to capture the fundamental dynamic properties of the initial system specification. In this way, AADS-T supports design space exploration by refinement of the AADL functionality and its implementation on an optimized platform.
Future work includes incorporation of AADS-T features that appear in V2.0 of the AADL standard.

8. ACKNOWLEDGEMENT

This work has been supported by ESTEC 22810/09/NL/JK HW-SW CODESIGN Project contracted to GMV Aerospace and Defence S.A.U.

The authors would like to acknowledge the aid received in this work from Francisco Ferrero Mateos and Elena Alaña Salazar of GMV, as well as from Juan Antonio de la Puente and Juan Zamorano.

9. REFERENCES

[1] SAE: AADL. June 2006, document AS5506/1. www.sae.org/technical/standards/AS5506/1.
[2] P. H. Feiler, D. P. Gluch, J. J. Hudak: The AADL: An Introduction. CMU. Pittsburgh. (2006).
[3] P. H. Feiler, J. J. Hudak: Developing AADL Models for Control Systems: Practitioner's Guide. CMU. 2006.
[4] SAE. Annex Behavior V2.0 AS5506, 2007.
[5] www.assert-project.net 2008 ESA/ESTEC.
[6] A. Burns et al.: The Ravenscar tasking profile for high integrity RT programs. Ada-Europe'98. Springer-Verlag.
[7] M. Perrotin et al.: The TASTE toolset: Turning human designed heterogeneous systems into computer built homogeneous software. ERTS2 2010, Toulouse, France.
[8] LEON2-FT ESA Microelectronics 2009 www.esa.int/TEC/Microelectronics/SEMUD70CYTE_0.html
[9] A.D. Pimentel et al.: A systematic approach to exploring embedded system architectures at multiple abstraction levels. IEEE Transactions on Computers, 2006.
[10] J. Hugues, B. Zalila, L. Pautet, F. Kordon: From the prototype to the final embedded system using the Ocarina AADL tool suite. ACM TECS, 2008. NY, USA.
[11] AADS V3.1 UC 2011. www.teisa.unican.es/AADS
[12] H. Posadas et al.: RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model. Design Automation for Embedded Systems. Springer. 2005.
[13] J. A. de la Puente et al.: The ASSERT VM: A Predictable Platform for Real-Time Systems. IFAC08. Korea.
[14] J. Zamorano et al.: The ASSERT VM kernel: Support for preservation of temporal properties. DASIA 2008. Spain.
[15] M. Bordin et al.: Automated Model-based Generation of Ravenscar-compliant Source Code. ECRTS05. Spain.
[16] S. Mazzini et al.: An MDE Methodology for the Development of High-Integrity RT Systems. DATE09.
[17] J. Kwon et al.: Ravenscar-Java: a High-Integrity Profile for Real-Time Java. ACM-ISCOPE, 2002. Washington.
[18] David C. Black, Jack Donovan: SystemC: From the ground up. Kluwer Academic Publishers. Boston (2004).
[19] SCoPE V1.1.0 UC 2009. www.teisa.unican.es/scope
[20] H. Posadas et al.: SystemC Platform Modeling for Behavioral Simulation and Performance Estimation of Embedded Systems. 2009. IGI Global. 978-1-60566750-8
[21] M. González: POSIX tiempo real. UC, Santander 2004.
[22] The Open Group: The Single UNIX Specification, V. 2, 1997. www.opengroup.org/onlinepubs/007908799.
[23] J. J. Labrosse: µC/OS RT Kernel. ISBN 0-87930-444-8.
[24] R. Varona Gómez, E. Villar: AADL Simulation and Performance Analysis in SystemC. ICECCS 2009. Germany.
[25] R. Varona Gómez, E. Villar: AADS+: AADL Simulation including the Behavioral Annex. ICECCS 2010. Oxford.
[26] P. H. Feiler, A. Greenhouse: OSATE Plug-in Development Guide. CMU. Pittsburgh. (2006).
[27] The Eclipse Foundation 2009. www.eclipse.org
[28] W3C: Extensible Markup Language (XML) W3C Recommendation (2006). www.w3.org/TR/REC-xml/
[29] http://hwswcodesign.gmv.com