Design and Implementation of Real-Time Distributed Systems with the ASSERT Virtual Machine*

Juan Zamorano, Juan A. de la Puente
Universidad Politécnica de Madrid (UPM)
E-28040 Madrid, Spain
jzamora@fi.upm.es, jpuente@dit.upm.es

Abstract

This paper describes how the ASSERT Virtual Machine, a dedicated execution platform that guarantees a predictable real-time behaviour, can be used to develop real-time embedded distributed systems with high-integrity requirements. The concept of virtual machine-level software components is developed, and sample programming patterns are provided. Components are used in the framework of the ASSERT development process, and are based on the Ravenscar Computational Model, a concurrent tasking model which can be statically analysed for a correct real-time behaviour. The current architecture of the ASSERT virtual machine is described in detail, and experience gained with its use in the last few years is provided.

1. Introduction

ASSERT (Automated proof-based System and Software Engineering for Real-Time systems)[1] was a collaborative research project partially funded by the European Commission under the 6th Framework Programme with the aim to propose new methods and tools for developing embedded real-time software. The project was carried out between 2004 and 2008 by a consortium of 28 partners, including space companies, research and academic laboratories, and software companies, led by the European Space Agency (ESA). Although the project focused on the aerospace domain, its results are also applicable to other domains with similar high-integrity, hard real-time requirements. The ASSERT results include a new development process for distributed embedded real-time systems, and a set of methods and tools supporting the process.

*This work has been partly funded by the Spanish Ministry of Science, project TIN2008-06766-C03-01 (RT-MODEL), the IST Programme of the European Commission under project IST-004033 (ASSERT), and the European Space Agency, ESTEC/Contract No. P1090383.

[1] `http://www.assert-project.net/`

The ASSERT process is based on a few basic principles:

- *Separation of concerns.* Functional aspects of a system are specified and implemented by system engineers using domain-oriented tools, and real-time aspects are specified in an abstract way and implemented by means of automatic tools.
- *Automatic code generation.* Development engineers work on high-level system models, and implementation code is automatically generated from such models. In particular, the code that implements the concurrency and real-time behaviour is automatically generated from high-level annotations to the models.
- *Property preservation.* Properties verified on high-level models are preserved through the development process and also at execution time.

This approach fits naturally into the Model-Driven Engineering (MDE) paradigm [17]. The ASSERT process is based on the integration of high-level, platform-independent models into different views, which are automatically transformed into implementation-oriented, platform-specific models and then into final code which can be compiled an run on an embedded computer platform. The transformations rules ensure that the properties of the models are preserved whenever possible, but for properties which cannot be statically guaranteed at development time (e.g. assumptions on arrival times of sporadic events or worst-case execution time), the execution platform must provide some means of monitoring and enforcing the required behaviour. Such an execution platform is called a *virtual machine* (VM), and is a fundamental component of any system that is required to exhibit a correct real-time behaviour at run time.

An instance of the ASSERT virtual machine was built during the ASSERT project [4], which has been later extended with additional support for communications and distributed systems. This paper focuses

978-1-4244-6850-8/10/$26.00 ©2010 IEEE

on the way the virtual machine concept can be used to develop high-integrity embedded real-time systems with guaranteed temporal properties. The paper is organized as follows: section 2 introduces the ASSERT software modelling and development process. The architecture of the virtual machine is described in section 3. The concept of virtual machine-level containers, which provides the basis for using the virtual machine functionality is elaborated in section 4. Some software tools supporting software development with the virtual machine are introduced in section 5. Finally, some conclusions based on current experience, as well as future work plans, are explained in section 6.

2. The ASSERT development process

The ASSERT software development process consists of the following four steps, which are carried out in an iterative way [13]:

1. *Modelling phase*, in which a system model is built using three different model views:

   (a) a *functional view*, which defines the sequential behaviour of the components of the system;

   (b) an *interface view*, which defines the ways that functional components are integrated and the interfaces for interaction between components;

   (c) a *deployment view*, which defines the physical components of the system, and the allocation of software components to physical resources.

   In MDE terms, the functional and interface views make up a *platform-independent model* (PIM) of the system, whereas the deployment view belongs at the *platform-specific model* (PSM) abstraction level.

2. *Model transformation.* A *concurrency view* is automatically generated from the above views. The concurrency view defines the concurrent and distributed architecture of the system in terms of the facilities provided by the underlying platform, e.g. threads, shared data, and messages.

   The concurrency view also belongs at the PSM level.

3. *Feasibility analysis.* Static analysis methods are used to verify that the required properties can be attained with the system architecture defined by the PSM views. The results of the analysis can be used to iterate on the physical architecture in order to improve the system behaviour.

4. *Code generation.* Source code is automatically generated from the concurrency view. The code can be compiled and deployed on the execution platform in order to get the final system implementation.

Figure 1 depicts the phases and model views of the ASSERT process.

System engineers, who are the final users of the process, have full control over the functional view. They can use different notations, as appropriate, to describe the functional behaviour of the system. Examples of such notations are Simulink®, SCADE®, SDL, different kinds of UML diagrams, and programming languages such as C/C++ or Ada. On the other side, non-functional properties such as temporal requirements (periods, deadlines, etc.), are ensured by restricting the user to set specific attributes on model components. To this purpose, interface view components are defined as *containers* into which the functional code is embedded in order to provide the building blocks of the system. The components of the interface view are called *application-level containers* (APLC), and are characterized by their provided and required interfaces that enable a complete system to be built as an aggregate of such components. Concurrency and real-time properties are defined in the interface view as attributes of ports.[2] For example, a port may have a *periodic(T)* attribute, which means that the associated service is executed every T seconds.

The components of the concurrency view are executable entities, which are automatically generated from the modelling phase views. They are called *virtual machine-level containers* (VMLC), as they directly rely on the concurrency, distribution, and real-time mechanisms provided by the VM. Functional elements are embedded in VMLC, which provide the elements required to implement the specified temporal and distribution behaviour. Figure 2 shows the relationships between APLC, VMLC, and the functional and deployment views.

An important feature of the ASSERT process is the adherence of the concurrency model, and thus of the VMLC, to the Ravenscar computational model (RCM)[2], 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 of shared data objects, protected by mutual exclusion synchronization. Accordingly, APLC must be designed in such a way that they can be implemented in terms of RCM-compliant VMLC. To this purpose, a formal grammar has been defined

[2] In UML a port is an interaction point between a component and its environment. Ports usually have one or more interfaces attached.

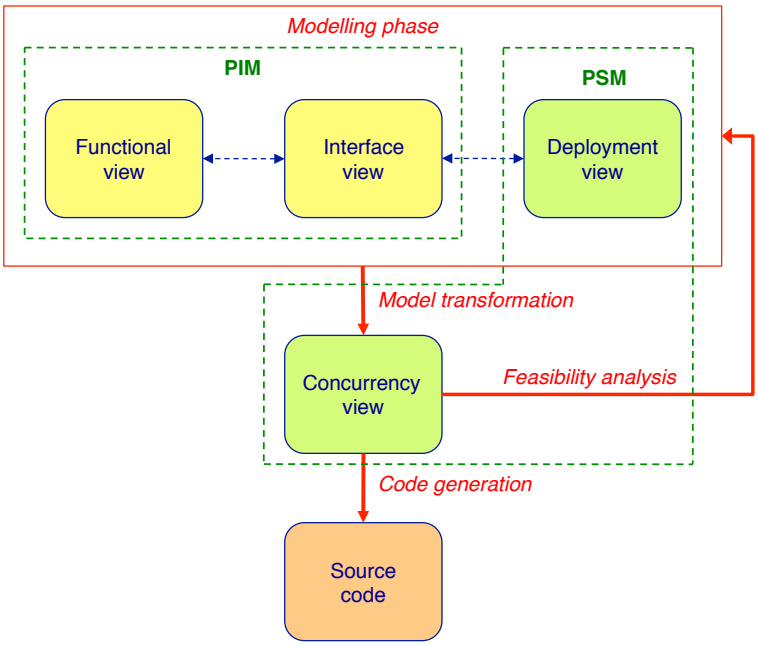

Figure 1. The ASSERT process phases.

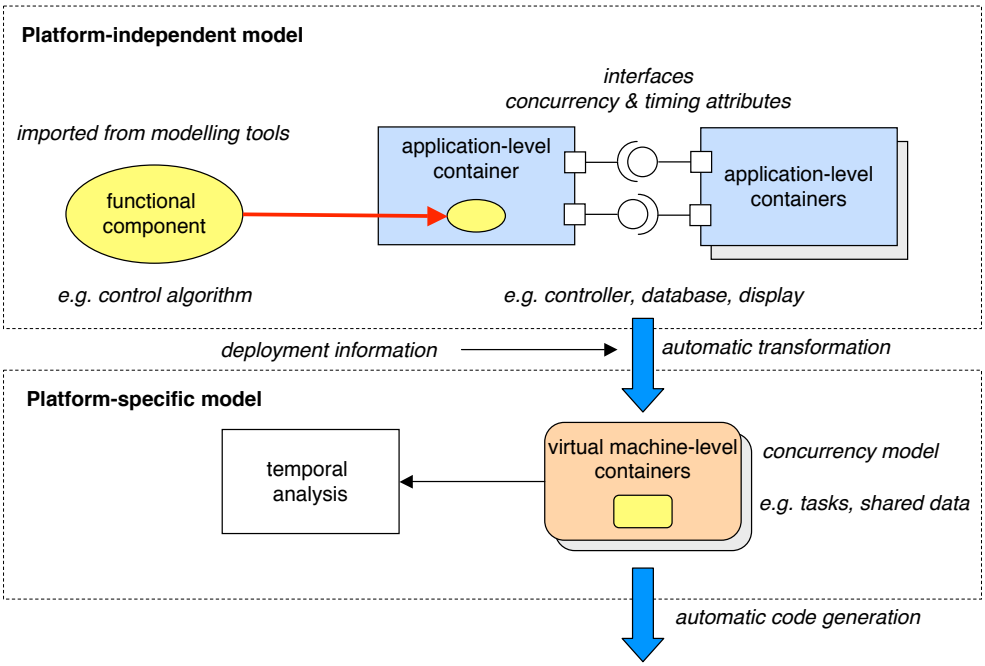

Figure 2. The ASSERT component model.

in order to specify a metamodel for application-level containers [1].

3. The ASSERT Virtual Machine

3.1. Definition and architecture

The ASSERT Virtual Machine (AVM)[4] is a dedicated execution environment supporting the ASSERT software process. In order to guarantee the non-functional properties, and especially the required temporal behaviour, the AVM implements static mechanisms for enforcing the required behaviour, whenever possible. To this purpose, the AVM only accepts for execution valid entities derived from the Ravenscar computational model, i.e. VMLC, as this is the basis that has been adopted for the feasibility analysis in the concurrency view. This approach requires that the implementation language and its associated tools are able to guarantee as many properties of the model as possible at compilation time. The Ada 2005 language [9] and the GNATforLEON compiler[3] have been chosen for the ASSERT VM as conformance to the Ravenscar profile can be statically checked by the compiler. The target hardware platform is the LEON processor family [5], a version of the SPARC v8 architecture [20] developed by ESA for on-board spacecraft computers.

System properties that cannot be statically checked are preserved by the virtual machine by monitoring and checking the real-time system behaviour at execution time. The Ada 2005 real-time mechanisms enable efficient run-time checks for some important temporal properties, such as deadlines, WCET budgets, and inter-arrival times for sporadic events [23].

The ASSERT Virtual Machine is composed of three main parts: real-time kernel, communication subsystem, and distribution middleware. Its architecture is depicted in figure 3. There is an instance of the AVM on every computer node in a distributed system.

3.2. Real-Time kernel

At the lower level of the AVM architecture is the real-time kernel, which implements basic concurrency and real-time mechanisms directly on top of the computer hardware. In order to comply with the requirement that the virtual machine only accepts run-time entities that exhibit the specified temporal behaviour, a Ravenscar-compliant kernel has been used. The AVM kernel is ORK+ [24], an extension of the original ORK [3] including real-time monitoring mechanisms. The kernel directly supports the RCM by implementing only static threads scheduled with a fixed-priority pre-emptive policy, as well a static mutual exclusion locks with immediate ceiling priority inheritance [18]. Such a kernel can be shown to provide a solid basis for analyzing the temporal behaviour of applications running on top of it [21].

[3] http://www.dit.upm.es/ork/.

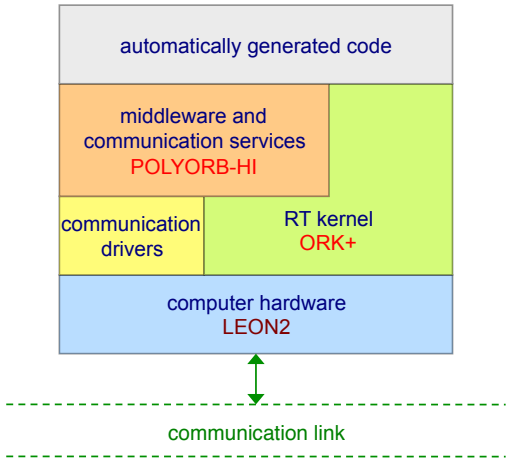

Figure 3. ASSERT Virtual Machine architecture.

3.3. Communication drivers

Communication drivers add basic communication facilities to the real-time kernel, and are a basis for implementing distributed systems. Indeed, in order to preserve the temporal properties of the applications using them, the drivers have to observe the Ravenscar profile restrictions and be integrated with the real-time kernel. The AVM kernel has been extended with communication drivers supporting generic asynchronous serial lines, as well as some other network interfaces which are common in the space domain, such as SpaceWire (ECSS-E-ST-50-12C)[4] and MIL-STD-1553B. The kernel structure makes it comparatively simple to develop drivers for other communication devices as well [10].

3.4. Middleware

The middleware layer provides distribution transparency to applications, so that application components need not take care of where other components reside. The main services provided by the middleware include name resolution, routing and transport of communications. These services are provided by PolyORB-HI,[5] a high-integrity derivative of the original PolyORB middleware [7]. PolyORB-HI is implemented as a library which can be tailored to each application by code-generation tools.

[4] http://spacewire.esa.int/
[5] http://aadl.enst.fr/polyorb-hi

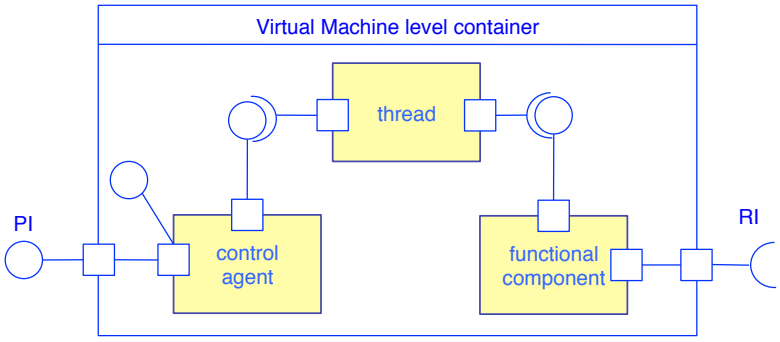

Figure 4. Internal structure of virtual machine-level containers.

4. Virtual machine level containers

Virtual machine-level containers (VMLC) are the only run-time entities that are accepted by the virtual machine. They are built from a reduced number of *archetypes* which ensure that their temporal behaviour is correct by construction. The containers provide concurrency, synchronization and timing mechanisms than enable the functional elements defined in the PIM to be embedded in a correct execution environment.

There are three kinds of VMLC:

- *Threaded containers* provide a periodic or sporadic execution environment for functions that are executed concurrently.
- *Protected containers* provide an execution environment for functions that operate on data that is shared by two or more threaded components. The container provides the synchronization mechanisms (locks) that ensure that all the functions exported to other components are executed in mutual exclusion.
- *Passive containers* have no concurrency or synchronization attributes and thus provide only a wrapper for purely functional behaviour. Passive components can be used by at most one threaded or protected component.

In order to simplify code generation, all VMLC archetypes are built from a common set of basic elements (figure 4):

- A *thread* for concurrent execution of the component functionality in the case of threaded containers;
- a *control agent*, providing synchronization mechanisms for threaded and protected containers;
- a *functional element*, embedding the functionality of the component.

Ada archetypes for these elements have been developed for every kind of virtual machine-level container [15]. Listing 1 shows, as an example, the archetype for a periodic thread including execution time budget monitoring.

Listing 1. Periodic thread with WCET overrun detection.

```
task body Cyclic_Thread is
   Next_Time : Time := <Start_Time>;
   Id : aliased constant Task_ID
          := Current_Task;
   WCET_Timer : Timer(Id'access); begin
   loop
     delay until Next_Time;
     Set_Handler(WCET_Timer,
                 Milliseconds(WCET_Budget),
                 WCET_Overrun_Handler);
      Functional.Cyclic_Operation;
      Next_Time := Next_Time
                   + Milliseconds(Period);
   end loop;
end Cyclic_Thread;
```

The code for the VMLC are generated from the concurrency view model using the archetypes. The middleware code is also automatically generated, and, although it is not based on VMLC archetypes, is made compliant with the Ravenscar computational model. Therefore, the temporal behaviour of all the code generated can be statically analysed.

5. Tool support

There are two variants of the ASSERT software process, which are supported by two different sets of tools:

- The *HRT-UML track* is based on HRT-UML/RCM, a UML profile using RCM as a meta-model [11]. This track uses UML as the prevalent notation for all model levels. An experimental toolset supporting this approach, based

on the Eclipse platform, was developed in the ASSERT project.[6]

- The *AADL track* is based on the Architecture Analysis and Design Language, an SAE standard for modelling embedded real-time systems [16]. AADL is used for all model views, and code is generated directly from the concurrency view model using the Ocarina tool[7] [8]. The TASTE toolset[8] is an open source toolset supporting this approach [14].

Both toolsets use response-time analysis tools to perform the feasibility analysis step of the ASSERT process. The TASTE toolset uses the Cheddar timing analysis tool [19], while the HRT-UML toolset includes an enhanced version of the MAST modelling suite[9] [6] for the same purpose. This allows the designer to verify the feasibility of the timing requirements on the platform-specific model, without having to deal with complex code details. If the analysis shows that the deployment of the system on a given platform does not fulfill the required timing behaviour, the designer can go back to the deployment view and choose an alternative platform with a different configuration and generate a new concurrency view which can be analysed again against the temporal requirements stated in the PIM interface view. This process can be iteratively repeated, and alternative platform configurations can be explored, until a satisfactory design is found.

Although the ASSERT toolsets do support an iterative, round-trip based design process for real-time systems, there are still some improvements that can be made. For one thing, response-time analysis tools require the worst-case execution time (WCET) of all of the system functions to be accurately estimated. While this a complex task when modern, complex processor architectures are used, recent advances enable efficient WCET estimations based on a hybrid approach including measurements as well as static analyisis [22]. Based on such results, our team is currently working at integrating the RapiTime tool[10] with the TASTE toolset, so that the execution time of the functional elements can be analysed directly on the concurrency view.

Another topic that deserves further exploring is temporal analysis of distributed systems. Although the Cheddar tool does not support distributed real-time analysis techniques, MAST includes support for some such methods [12]. The authors' team are working on integrating MAST with the TASTE toolset in order to extend the timing analysis capabilities of the toolset to distributed systems as well.

[6] `http://www.math.unipd.it/~tullio/Research/ASSERT/Tutorial/`.

[7] `http://ocarina.enst.fr/`

[8] `http://www.semantix.gr/assert/`

[9] `http://mast.unican.es/`

[10] `http://www.rapitasystems.com/rapitime`

6. Conclusions and future work

The ASSERT software process and its associated tools open the way to a new approach to the development of software intensive automation systems. The model-driven engineering paradigm fits naturally into the cultural framework of automation engineers, who customarily use high-level models to design and reason about control algorithms, event-driven control methods, and other control-related aspects of automation systems. The ASSERT approach shows that MDE can successfully applied to the development of complex, distributed real-time systems, letting systems engineers concentrate of the application-oriented aspects of the system under development, and automatically generating "glue" code for the concurrency, real-time, and distribution aspects based on high-level specifications and computing platform descriptions. Other non-functional characteristics, such as safety or dependability, can also be analysed at the platform-specific modelling level.

The CHESS project[11] seeks to widen the application areas and property analysis of MDE for embedded real-time systems. Other lines of future work include extending MDE to include hardware-software co-design methods, and integrating it with platform simulation tools.

References

[1] M. Bordin and T. Vardanega. Correctness by construction for high-integrity real-time systems: A metamodel-driven approach. In N. Abdennadher and F. Kordon, editors, *12th International Conference on Reliable Software Technologies — Ada-Europe 2007*, number 4498 in LNCS, pages 114–127. Springer-Verlag, 2007.

[2] A. Burns, B. Dobbing, and G. Romanski. The Ravenscar tasking profile for high integrity real-time programs. In L. Asplund, editor, *Reliable Software Technologies — Ada-Europe'98*, number 1411 in LNCS, pages 263–275. Springer-Verlag, 1998.

[3] J. A. de la Puente, J. F. Ruiz, and J. Zamorano. An open Ravenscar real-time kernel for GNAT. In H. B. Keller and E. Plödereder, editors, *Reliable Software Technologies — Ada-Europe 2000*, number 1845 in LNCS, pages 5–15. Springer-Verlag, 2000.

[4] J. A. de la Puente, J. Zamorano, J. A. Pulido, and S. Urueña. The ASSERT Virtual Machine: A predictable platform for real-time systems. In M. J. Chung and P. Misra, editors, *Proceedings of the 17th IFAC World Congress*. IFAC-PapersOnLine, 2008.

[5] Gaisler Research. *LEON2 Processor User's Manual*, 2005.

[11] `https://www.artemisia-association.org/chess`

[6] M. González Harbour, J. J. Gutiérrez, J. C. Palencia, and J. M. Drake. MAST modeling and analysis suite for real time applications. In *Proceedings of 13th Euromicro Conference on Real-Time Systems*, pages 125–134, Delft, The Netherlands, June 2001. IEEE Computer Society Press.

[7] J. Hugues, L. Pautet, and B. Zalila. From MDD to full industrial process: Building distributed real-time embedded systems for the high-integrity domain. In *Composition of Embedded Systems. Scientific and Industrial Issues*, number 4888 in LNCS, pages 35–52. Springer, 2007.

[8] J. Hugues, B. Zalila, L. Pautet, and F. Kordon. From the prototype to the final embedded system using the Ocarina AADL tool suite. *ACM Tr. Embedded Computer Systems*, 7(4):1–25, 2008.

[9] ISO/IEC. *Std. 8652:1995/Amd 1:2007 — Ada 2005 Reference Manual. Language and Standard Libraries*, 2007. Published by Springer-Verlag, ISBN 978-3-540-69335-2.

[10] J. López, Ángel Esquinas, J. Zamorano, and J. A. de la Puente. Experience in programming device drivers with the Ravenscar profile. *Ada User*, 31(2), June 2010.

[11] S. Mazzini, S. Puri, and T. Vardanega. An MDE methodology for the development of high-integrity real-time systems. In *Design, Automation and Test in Europe, DATE 2009*, pages 1154–1159. IEEE, 2009.

[12] J. C. Palencia Gutiérrez and M. González Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *RTSS 1999: Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 328–339, December 1999.

[13] M. Panunzio and T. Vardanega. A metamodel-driven process featuring advanced model-based timing analysis. In N. Abdennadher and F. Kordon, editors, *12th International Conference on Reliable Software Technologies — Ada-Europe 2007*, number 4498 in LNCS, pages 128–141. Springer-Verlag, 2007.

[14] M. Perrotin, E. Conquet, P. Dissaux, T. Tsiodras, and J. Hugues. The TASTE toolset: Turning human designed heterogeneous systems into computer built homogeneous software. In *5th Int. Congress on Embedded Real-Time Software and Systems — ERTS2 2010*, May 2010.

[15] J. Pulido, J. A. de la Puente, M. Bordin, T. Vardanega, and J. Hugues. Ada 2005 code patterns for metamodel-based code generation. *Ada Letters*, XXVII(2):53–58, August 2007. Proceedings of the 13th International Ada Real-Time Workshop (IRTAW13).

[16] SAE. *Architecture Analysis and Design Language (AADL) — AS5506A*, January 2009. Available at `www.sae.org`.

[17] D. C. Schmidt. Model-driven engineering. *IEEE Computer*, 39(2), 2006.

[18] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Tr. on Computers*, 39(9), 1990.

[19] F. Singhoff, A. Plantec, and P. Dissaux. Can we increase the usability of real time scheduling theory? the Cheddar project. In *Ada-Europe '08: Proceedings of the 13th Ada-Europe international conference on Reliable Software Technologies*, pages 240–253, Berlin, Heidelberg, 2008. Springer-Verlag.

[20] SPARC International. *The SPARC architecture manual: Version 8*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[21] T. Vardanega, J. Zamorano, and J. A. de la Puente. On the dynamic semantics and the timing behaviour of Ravenscar kernels. *Real-Time Systems*, 29(1):1–31, 2005.

[22] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):1–53, 2008.

[23] J. Zamorano, J. A. de la Puente, J. Hugues, and T. Vardanega. Run-time mechanisms for property preservation in high-integrity real-time systems. In *OSPERT 2007 — Workshop on Operating System Platforms for Embedded Real-Time Applications*, Pisa. Italy, July 2007.

[24] J. Zamorano, J. A. de la Puente, J. A. Pulido, and S. Urueña. The ASSERT virtual machine kernel: Support for preservation of temporal properties. In *Data Systems in Aerospace —— DASIA 2008*, Palma de Mallorca, Spain, 2008.