Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

Mixed-criticality design of a satellite software system [1]

Emilio Salazar * **Alejandro Alonso** * **Jorge Garrido** *

* *Universidad Politécnica de Madrid (UPM), Spain*
E-mail: {esalazar, aalonso}@dit.upm.es, jgarrido@datsi.fi.upm.es

Abstract:
The continuous increment of processors computational power and the requirements on additional functionality and services are motivating a change in the way embedded systems are built. Components with different criticality level are allocated in the same processor, which give rise to mixed-criticality systems. The use of partitioned systems is a way of preventing undesirable interferences between components with different criticality level. An hypervisor provides these partitions or virtual machines, ensuring spatial, temporal and fault isolation between them. The purpose of this paper is to illustrate the development of a mixed-critical system. The attitude control subsystem is used for showing the different steps, which are supported by a toolset developed in the context of the MultiPARTES research project

Keywords: Real-time systems, Partitioned Systems, Mixed Criticality, Model Driven Engineering

1. INTRODUCTION

The continuous increment on processors computational power and the requirements on additional functionality and services are motivating a change in the way embedded systems are built. Complex systems, composed by a set of interacting computers with demanding applications will be commonplace. These systems will include applications of different nature. In particular, components with different criticality level will coexist in the same system. The integration of a large number of functionalities in the same execution platform poses a number of new technical challenges.

The development of these safety systems must follow thorough and methods, usually conforming to standards, such as those for electronic systems (IEC 61508), airborne civil avionics (DO-178B), European railway (EN 50128]), or European space (ECSS). Most of these standards assigns integrity or criticality levels to the different components of the system. These levels represent the likelihood of a safety-related system satisfactorily performing the required safety functions under all de stated conditions within a stated period of time. The integrity level determines the development methods and validation and verification techniques to be used. In some domains, it is necessary to certificate the system for ensuring it can be trusted, i.e. it will operate in a safe and secure way for persons and the environment. The certification is the process of providing evidence to a regulation entity that the system will behave as expected.

Traditionally, software components with different criticality level were located on different computers, ensuring that there will no undesirable interferences. However, currently these approach is not advisable. Available processors have enough processing capacity for running the whole system. In addition, there are requirements on size, weight, and power that are not fulfilled if several computers are used. It is more suitable to allocate the whole system in the same computer. This implies the co-existence of applications with different criticality level. These are named mixed-criticality systems. In order to meet the authorities requirements, it means that the whole system has to be certified, even the non-critical components. This implies prohibitive costs.

A suitable approach relies on the use of an hypervisor that provides partitions or virtual machines. The hypervisor ensures spatial, temporal and fault isolations. In this way, it is possible execute applications with different criticality level on different partitions, precluding undesirable interferences.

The aim of this paper is to illustrate the development process of mixed-criticality systems using the tools and methods developed in the context of the MultiPARTES [2] project (Multi-cores Partitioning for Trusted Embedded Systems, Trujillo et al. (2013)). The case study is based on the Attitude Control of the UPMSAT2 satellite. The original design is modified to include two separate partitions, assuming that they have different criticality level.

The authors are participating in the development of the UPMSAT2, which is an satellite developed by the University for research and teaching purposes. This paper describes a case study based on the ADCS (Attitude Data Control System), which is part of the OBDH (On-Board Data Handling). This paper illustrates how this subsystem is developed as a partitioned mixed-criticality system.

[1] This work has been partially funded by the Spanish Government, project HI-PARTES (TIN2011-28567-C03-01), and by the European Commission FP7 programme project MultiPARTES (IST 287702).

[2] www.multipartes.eu

2. MULTIPARTES APPROACH

2.1 Toolset overview

The development of partitioned mixed-criticality embedded systems poses new challenges, such as the decision on the number of partitions, the applications and resources assigned to each of them, the validation of the system, etc. In order to support the developer on these activities a toolset is being developed in the context of the MultiPARTES project.

This framework is based on Model Driven Engineering (MDE) (Schmidt, 2006), that is a software development approach managed by the Object Management Group (OMG), that allows engineers to raise the abstraction level of the languages and tools used in the development process. It also helps designers to isolate the information and processing logic from implementation and platform aspects. A basic objective of MDE is to put the model concept on the critical path of software development. This notion changes the previous situation, turning the role of models from contemplative to productive.

The main components of the toolset and data flows are depicted in figure 1. Their basic role are:

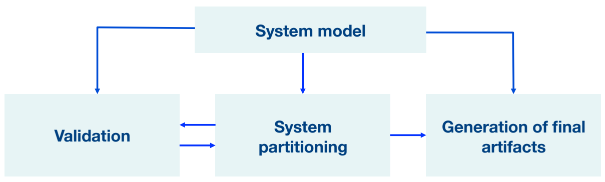


Fig. 1. Overall toolset architecture

- *System modeling:* It comprises the main input to the tool. It is composed by three models for describing the execution platforms, the applications, and the restrictions to be applied in the partitioning. Models can be enriched with information on non-functional requirements, such as real-time or safety.
- *Partitioning tool*: It is in charge of generating a system partitioning, that is represented by the *deployment model*, which defines system partitions, the assignment of applications to partitions, and the characteristics of the partitions, including the operating system, processor time, memory, etc. The partitioning tool takes as input the system model. It has to consider information, such as the applications' criticality level, their required operating system and hardware devices, etc. Based on this information it generates a deployment model that meets the restrictions and some basic requirements.
- *Validation*: Full correctness of a system partitioning may require complex checks that are difficult to integrate within a single tool. In addition, it is desirable for the toolset to be extended for supporting additional non-functional requirements. Then it is convenient to be able to use *Validation tools* that checks the correctness of the system configuration with respect to a given criteria. As instance, a response time analysis tool can be used to ensure deadlines fulfillment for real-time applications.
- *Generation of final artifacts*: when the system partitioning is correct a number of transformation tools generates a set of outcomes that are necessary for creating and building the final system. Information on non-functional requirements is used for this generation. If an application is labelled as critical, the generated code will meet the coding guidelines for this type of systems.

Separation of concerns is one of the main design principles in MDE. It implies keeping different concepts in different sections. In the case of the MultiPARTES methodology, this principle has been applied into the system modeling. In this context, there are different aspects that are conceptually independent one each other and thus, they are described in separated models:

- Application model (described in section 3).
- Platform model (described in section 4).
- Partitioning restriction model (described in section 5).

A major goal of this approach is maximizing the reusability of the models. There can exist a number of applications and platforms. They can be combined to build a new systems, without having to modify the original models.

Another important goal of this strategy is converting a non-partitioned system into a partitioned systems, without modifying these models. It is possible as well, modifying the partitioning schema without having to change neither the platform nor the application model. In both cases, the only impacted model is the partitioning restriction model.

2.2 Attitude Control System of the UPMSat-2

The UPMSat-2 satellite can be physically described as a micro-satellite, with a mass of approximately 50kg. The payload of the satellite is a set of different experiments proposed by both industrial companies and research groups. These experiments are focused on experience acquisition and testing in the space environment. The satellite is planned to be launched in 2015, and its expected operating period is two years.

The UPMSat-2 has an on-board computer (OBC), which executes the on-board software (OBSW). One of the OBSW's subsystems, among others, is the Attitude Data Control (ADCS). The ADCS subsystem is in charge of keeping the satellite in a position that the radio antenna is always visible from the Earth.

The UPMSat-2 ADCS is used in this research group for demonstration and validation of oen developed technologies. In this paper, it is described its development using the MultiPARTES methodology and techniques. The most relevant feature of the strategy is the use of the mixed-criticality (Burns and Davis, 2013; European Comission) partitioned (Dobbing, 2000; Rushby, 1999) architecture integrated in a model-driven development process. The original ADCS subsystem design has been modified assuming that there are two separated components with different criticality levels and that cannot coexist in the same partition:

- Attitude control algorithm component (`ADCS_Control`). It is the component with the mathematical attitude determination model. It is modeled with Simulink (Mathworks, 2013). This tool automatically generates the code that implements the algorithm.

- Input/Output component (ADCS_IO). The ADCS subsystem requires accessing two hardware elements: the magnetorque and the magnetometer. This component holds the drivers and the logic required for accessing these devices.

In a traditional approach, these components would execute in two different computers, since two different criticality level applications could not run on the same processor. Nevertheless, in the mixed-criticality approach proposed in MultiPARTES, a partitioned system is used instead of adding more hardware.

A partitioned system adds an extra software layer between the hardware and the business logic called hypervisor (Hartner and Gerstinger, 2008). The goal of the hypervisor is virtualizing the available hardware in such a way that they can be shared by different software components as if each component ran its own hardware.

Each piece of software that runs isolated is called partition. A partition can be from a general-purpose operating system running applications (e.g. Linux) until a bare-board software.

Aside from the aforementioned properties, a safety real-time hypervisors (e.g. XtratuM (Masmano et al., 2005, 2009; Esquinas et al., 2011)) ensures that all safety, security and real-time properties of each partition are met as well, which makes possible that safety-critical software share hardware resources.

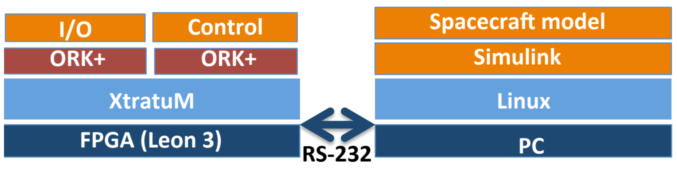


Fig. 2. ADCS architecture overview

In our example, there are two component with different criticality levels running on the on-board computer. Because of their criticality levels, they cannot run on the same partition. For this reason, there are two ORK+ (de la Puente et al., 2008) partitions (see figure 2). While one partition is executing the control algorithm, the other one runs the I/O logic.

2.3 Demonstrator debugging: Hardware-in-the-loop

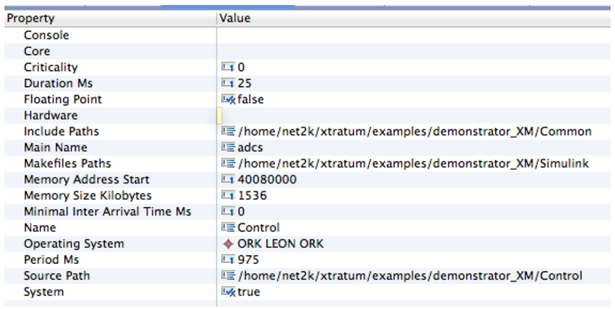
For testing the control component a hardware-in-the-loop (HIL) approach is used. The goal of this technique is adding the complexity of the plant under the control of the testing platform.

On the one hand, a PC executes a Simulink model, which simulates the behavior of satellite taking in account the actions performed in the magnetorque and providing coherent reads on the magnetometer. On the other hand, a LEON3 (Gaisler Research) FPGA executes the control software that it is being tested. The communication between the FPGA and the PC is performed with a RS-232 serial port.

3. APPLICATION MODEL

The aim of this model is to provide a functional description of the application, and the required information for the operations to be performed by the toolset. The functional description is usually provided as UML 2.2 (OMG, 2011) class models. These are enriched with annotations useful for system partitioning and deployment, which includes:

- Criticality level: essential information about the application is its criticality level. This information is crucial for activities such as partitioning, validation, or code generation. For instance, the partitioning tool cannot allocate two applications with different criticality level in the same partition.
- Resource needs: the application has to describe the hardware resources needed for their execution, such as CPU time, input/output devices, etc. This information is annotated using the UML-MARTE profile (OMG 2011b).
- Operating system. The operating system required by the application.
- Configuration information: It is needed to provide the information that is required for the configuration of the hypervisor, and for the generation of the final system. This information includes: i) partition flags, for configuring characteristics such as the use of floating point arithmetic, ii) start memory addresses, it specifies where will be the application allocated on the partition memory, although this information is optional, as can be automatically determined, iii) memory size for the application, iv) information about the location of the code, the compiler, and the required libraries, is provided for building the final system. Figure 3 shows an example of the configuration information for an application.



Property	Value
Console	
Core	
Criticality	0
Duration Ms	25
Floating Point	false
Hardware	
Include Paths	/home/net2k/xtratum/examples/demonstrator_XM/Common
Main Name	adcs
Makefiles Paths	/home/net2k/xtratum/examples/demonstrator_XM/Simulink
Memory Address Start	40080000
Memory Size Kilobytes	1536
Minimal Inter Arrival Time Ms	0
Name	Control
Operating System	ORK LEON ORK
Period Ms	975
Source Path	/home/net2k/xtratum/examples/demonstrator_XM/Control
System	true

Fig. 3. Sample of the application configuration information

Functional models can be enriched with annotations related with non-functional requirements. The toolset currently supports safety and real-time annotations. The use case in this paper uses real-time annotations. Real-Time information is provided for validation and code generation purposes. It includes the description of the activation pattern, deadlines, or computation time. This information is annotated in the functional model using the UML-MARTE standard.

Figure 4 depicts a simplified version of the ADCS subsystem class diagram.

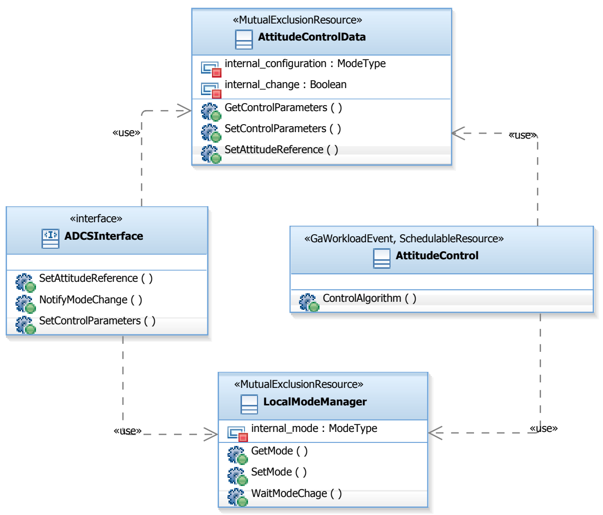
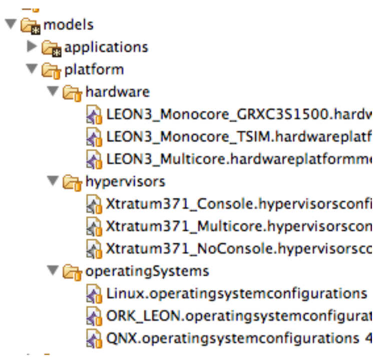


Fig. 4. Detail of the ADCS subsystem UML class diagram

4. PLATFORM MODEL

The execution platform is characterized by the hardware, operating system and the hypervisor used in the system. These elements are modelled separately, in order to facilitate its reuse. In this way, it is possible for a developer that has already defined a hardware and a hypervisor model to add a number of operating systems, without having to modify the former models. Figure 5 shows the components of the platform model that are available in the use case development framework.

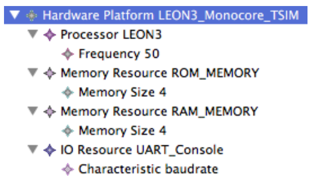


```
▼ models
  ▶ applications
  ▼ platform
    ▼ hardware
        LEON3_Monocore_GRXC3S1500.hardv
        LEON3_Monocore_TSIM.hardwareplatf
        LEON3_Multicore.hardwareplatformm
    ▼ hypervisors
        Xtratum371_Console.hypervisorsconfi
        Xtratum371_Multicore.hypervisorscon
        Xtratum371_NoConsole.hypervisorscc
    ▼ operatingSystems
        Linux.operatingsystemconfigurations
        ORK_LEON.operatingsystemconfigurat
        QNX.operatingsystemconfigurations 4
```

Fig. 5. Platform models library

The components of a platform model are:

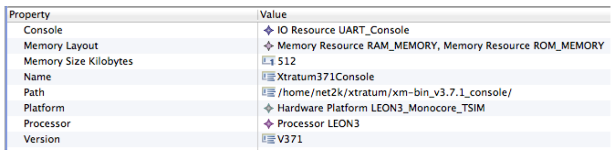
- Hardware: This model is used for describing a hardware system. The most important elements modelled are the processor, its cores (if any), the memory layout and the I/O devices. It is also possible to model additional hardware devices, such as buses, FPGAs, etc. Figure 6 shows the hardware platform used in the UPMSat-2 use case. It defines a LEON3 processor at 50 MHz with 4 MB RAM, 4 MB ROM, and an UART device.
- Hypervisor: Each hypervisor has its own configuration parameters, which are stored in this model. Figure 7 shows the configuration of the XtratuM hypervisor used in this case study. The main features represented in this model are: the hardware where the hypervisor is running, the path to the XtratuM



```
▼ Hardware Platform LEON3_Monocore_TSIM
  ▼ Processor LEON3
      Frequency 50
  ▼ Memory Resource ROM_MEMORY
      Memory Size 4
  ▼ Memory Resource RAM_MEMORY
      Memory Size 4
  ▼ IO Resource UART_Console
      Characteristic baudrate
```

Fig. 6. Use case hardware model

libraries, its version, and the hypervisor memory layout.
- Operating system. The last modelled element of the platform is the operating system. There is a wide range of operating systems, and each one has its specific configuration parameters. Different partitions may hold different operating systems. This model is designed to be as flexible and extensible as possible. Figure 8 shows how is the ORK+ represented in the platform model. This model contains general operating system information such as its name, criticality support, or maximum number of tasks that can be run in parallel. It also contains specific data of ORK+, such as the path of Ada compiler, the path to the drivers sources, the debug flag, etc.



Property	Value
Console	IO Resource UART_Console
Memory Layout	Memory Resource RAM_MEMORY, Memory Resource ROM_MEMORY
Memory Size Kilobytes	512
Name	Xtratum371Console
Path	/home/net2k/xtratum/xm-bin_v3.7.1_console/
Platform	Hardware Platform LEON3_Monocore_TSIM
Processor	Processor LEON3
Version	V371

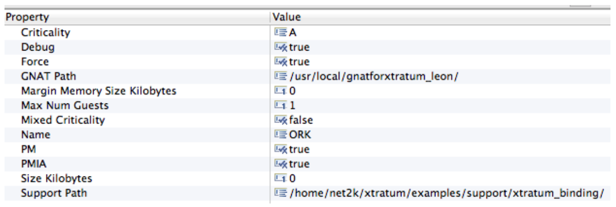
Fig. 7. Use case XtratuM model

5. PARTITIONING RESTRICTIONS MODEL

Application and platform models are intended to be independent of a particular system, in order to be easily reused. The partitioning restriction model is aimed at relating platform elements and applications, for a given system. In this way, it is possible to state relations between an application, and the operating system to be used or a particular hardware device that is required for a particular system, without having to modify these elements.

The allocation of applications to partitions is a particular type of the general problem of the resource allocation, which is NP-hard. However, there are allocations that are not acceptable for a particular system. Partitioning restrictions are also used for trimming the solution space, and ensuring that the proposed partitioning is suitable. A design goal for the partitioning restrictions model was to keep it simple for two main reasons:

- Algorithm performance: the simpler the rules, the faster will be the generation of a system partitioning.



Property	Value
Criticality	A
Debug	true
Force	true
GNAT Path	/usr/local/gnatforxtratum_leon/
Margin Memory Size Kilobytes	0
Max Num Guests	1
Mixed Criticality	false
Name	ORK
PM	true
PMIA	true
Size Kilobytes	0
Support Path	/home/net2k/xtratum/examples/support/xtratum_binding/

Fig. 8. Use case ORK+ model

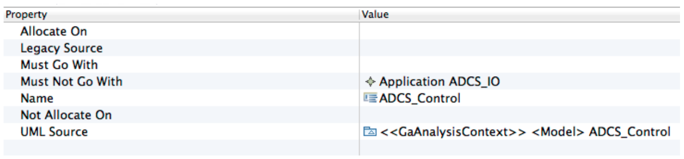
Most of the high abstraction level restrictions can be translated to constraints of the form *application must (not) execute with* or *application must (not) be allocated on.*

- Facilitate the automatic generation of restrictions: the application model can be extended with the specification of non-functional requirements. The automatic generation of restrictions is a way of ensuring that a system partitioning is compliant with such specification.

Currently, the following types of restrictions are supported:

- Application must (not) run with an application. This restriction states whether two applications can (not) be allocated in the same partition. This type of restrictions can be generated automatically. For example, applications with different criticality level cannot coexist int he same partition.
- Application must (not) be allocated on a partition. This restriction ensures that in the resulting partitioning schema the provided application will (not) be allocated on the given partition.

Figure 9 shows a restriction, where the user explicitly forbids for the application `ADCS_IO` to be allocated in the same partition than the application `ADCS_Control`. It will be taken into account by the partitioning tool.



Property	Value
Allocate On	
Legacy Source	
Must Go With	
Must Not Go With	Application ADCS_IO
Name	ADCS_Control
Not Allocate On	
UML Source	<<GaAnalysisContext>> <Model> ADCS_Control

Fig. 9. Partitioning restrictions for an application

The partitioning tool is in charge of generating a deployment model, which defines a system partition that meets the requirements stated in the restrictions. The resulting system can be validated, using external tools. A response time analysis tool could be used for checking whether time requirements are met, given the current system configuration.

6. SATELLITE ARTEFACTS GENERATION

The last step in the MultiPARTES tool chain is the artefacts generation (see figure 10). Once the system has been modelled and processed by the tools, a valid deployment is generated. The relevant information for artefacts generation is extracted on a simplified model, that is called neutral. This step facilitates the development of transformation tools for different programming languages, and adds modularity to the framework.

Model-to-model transformations has been developed using Query-View-Transformation Operational Language (QVT), the OMG standard for this kind of transformations. On the other hand, model-to-text generators has been developed with the OMG standard Model to Text Language (MTL) in its Acceleo's implementation[3]. The following artefacts are automatically generated:

[3] `http://www.eclipse.org/acceleo/`

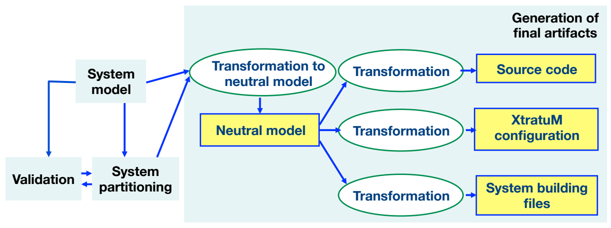


Fig. 10. MultiPARTES generation workflow

- XtratuM configuration files: The required files for executing the applications in the XtratuM hypervisor are generated. The main outcome is the XtratuM XML configuration file, which describes the partitions, their hardware resources, partitions scheduling, and the allocation of applications to partitions. Some details are shown in the listing 1. It describes two partitions, a major frame with a period of 300ms, and two slots with a duration of 200ms and 100ms. This information defines the execution timeframe of each of the partitions. The memory addresses that are shown are extracted from the platform model. This data is obtained from the application and platform models.

```
<xtratum:SystemDescription xmlns:xtratum="http://www.xtratum.org/xm-3.x" name="S">
  <xtratum:HwDescription>
    <xtratum:MemoryLayout>
      <xtratum:Region size="4MB" start="0x0" type="rom"/>
      <xtratum:Region size="4MB" start="0x40000000" type="stram"/>
    </xtratum:MemoryLayout>
    <xtratum:ProcessorTable>
      <xtratum:Processor frequency="50MHz" id="0">
        <xtratum:CyclicPlanTable>
          <xtratum:Plan id="0" majorFrame="300ms" name="Plan0">
            <xtratum:Slot duration="200ms" id="0" partitionId="1" start="0ms"/>
            <xtratum:Slot duration="100ms" id="1" partitionId="0" start="200ms"/>
          </xtratum:Plan></xtratum:CyclicPlanTable></xtratum:Processor>
      </xtratum:ProcessorTable>
    <xtratum:Devices>
      <xtratum:Uart baudRate="115200" id="0" name="UART_Console"/>
    </xtratum:Devices>
  </xtratum:HwDescription>
  <xtratum:XMHypervisor console="UART_Console">
    <xtratum:PhysicalMemoryArea size="512KB"/>
  </xtratum:XMHypervisor>
      . . . . .
</xtratum:SystemDescription>
```

Listing 1: *Xtratum configuration file two partitions*

Two additional files are generated: a linker file, with information on how to link each partition, anda a generic boot assembler, for setting a valid stack and jumps for executing the application code.

- System makefiles. Aside from the XtratuM configuration files, it is also generated the required files for compiling and build the whole system, including the applications. A makefile for each partition is created. In this makefile is compiled the application against its required libraries and operating system. Then it is built and assembled the partition. It is also generated a global makefile that is in charge of assembling all partitions in the final XtratuM binary package.
- Source code skeletons: the toolset can generate code skeletons, according to the application model. In this use case, Ada 2005 skeletons are generated. In order to be suitable for high criticality systems, the concurrency model is based on the Ravenscar profile (Burns, 1999), and the code follows the recommendations in

(HRG, 1998), avoiding the use of dynamic memory or object orientation features. Three different types of components are generated: i) active: they have a proper execution flow, so are implemented as Ada tasks, which can be activated by a clock (periodic) or an event (sporadic), ii) protected: these are a kind of monitors for ensuring synchronization and mutual exclusion between tasks, iii) passive: these components have no real-time stereotypes and they hold the auxiliar code invoked from the active components.

Listing 2 shows the code generated for a periodic task, which is in charge of executing the control algorithm, which is encapsulated in the `AttitudeControl` procedure. Internally, it includes a call to the `ADCS_IO` partition for getting the magnetometer data, the execution of the control algorithm, and another call to `ADCS_IO` for providing the data for the actuator.

```
task body HKPeriodicTask_Type is
        Interval   : constant Time_Span := Milliseconds (2000);
        Next_Time : Time                 := Clock + Milliseconds (0);
        Canceled   : Boolean             := False;
begin
        delay until Next_Time; -- Offset
        Next_Time := Clock + Interval;
        loop
                -- User defined code starts here
        AttitudeControl;
                -- User defined code finishes here
                delay until Next_Time; -- Wait until next activation
                Next_Time := Next_Time + Interval;
        end loop;
end HKPeriodicTask_Type;
```

Listing 2: *Code skeleton of a periodic task*

7. CONCLUSIONS

This paper has described the development of a mixed-criticality system. The case study has been the attitude control of the UPMSAT-2. This system has been split on two partitions: one where the control algorithm executes, the other where the I/O subsystem is executed. Both partitions interact for interchanging sensors and actuators information. The system has been developed with the toolset developed in the context of the MultiPARTES project. It includes tools for modelling the system, generating a system partitioning, and generating the final artefacts, for simplifying the final system deployment. The aim has been to validate the overall approach, to illustrate the generation of these type of systems, and to show a toolset with some of the required support for mixed-criticality systems. The final system was developed successfully. The configuration files for XtratuM where valid and the system was fully built issuing only one command.

REFERENCES

(HRG 1998). *Guide for the use of the Ada Programming Language in High Integrity Systems.* ISO. Working draft ISO/IEC/JTC1/SC22/WG9 (Ada) / HRG. Version 3.3.

Burns, A. (1999). The Ravenscar profile. *Ada Letters*, XIX(4), 49–52.

Burns, A. and Davis, R. (2013). Mixed criticality systems — A review. Technical report, University of York. URL `http://www-users.cs.york.ac.uk/~burns/review.pdf`.

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

Dobbing, B. (2000). Building partitioned architectures based on the Ravenscar profile. *Ada Lett.*, XX(4), 29–31. doi:http://doi.acm.org/10.1145/369264.369266.

Esquinas, A., Zamorano, J., de la Puente, J.A., Masmano, M., Ripoll, I., and Crespo, A. (2011). ORK+/XtratuM: An open partitioning platform for Ada. In A. Romanovsky and T. Vardanega (eds.), *Reliable Software Technologies — Ada-Europe 2011*, number 6652 in LNCS, 160–173. Springer-Verlag.

European Comission (2012). Mixed criticality systems.

Gaisler Research (2012). *LEON3 - High-performance SPARC V8 32-bit Processor. GRLIB IP Core User's Manual.*

González Harbour, M., Gutiérrez, J.J., Palencia, J.C., and Drake, J.M. (2001). MAST modeling and analysis suite for real time applications. In *Proceedings of 13th ECRTS.* IEEE Computer Society Press, Delft, The Netherlands.

Hartner, G. and Gerstinger, A. (2008). Safety supervision layer. In *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, 252 –257. doi: 10.1109/INDIN.2008.4618104.

Masmano, M., Ripoll, I., and Crespo, A. (2005). An overview of the XtratuM nanokernel. In *OSPERT 2005 — Workshop on Operating System Platforms for Embedded Real-Time Applications.* Palma de Mallorca.

Masmano, M., Ripoll, I., Crespo, A., and Metge, J.J. (2009). XtratuM: a hypervisor for safety critical embedded systems. In *11th Real-Time Linux Workshop.* Dresden. Germany.

Mathworks (2013). *Simulink.* URL `www.mathworks.com/products/simulink`.

MOF (2005). *ISO/IEC 19502:2005 Information technology – Meta Object Facility (MOF).* ISO.

MTL (2008). *MOF Model to Text Transformation Language (MOFM2T).* OMG.

OMG (2011). *Unified Modeling Language (UML).* URL `http://www.omg.org/spec/UML/2.4.1/`. Version 2.4.1.

OMG 2011b (2011). *OMG UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems.* URL `http://www.omg.org/spec/MARTE/`. Version 1.1.

QVT (2011). *Meta Object Facility (MOF) 2.0 Query/View/Transformation.* OMG.

Rushby, J. (1999). Partitioning for safety and security: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347.

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

Trujillo, S., Crespo, A., and Alonso, A. (2013). MultiPARTES: Multicore virtualization for mixed-criticality systems. In *Euromicro Conference onDigital System Design, DSD 2013*, 260–265. doi:10.1109/DSD.2013.37.