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

Model Driven Design Support for Mixed-Criticality Distributed Systems

A. Armentia*, A. Agirre, E. Estévez***, J. Pérez**, M. Marcos***

** Dept. Ingeniería de Sistemas y Automática. ETSI Bilbao, UPV/EHU*
Spain (e-mail: @ehu.es).
*** Ikerlan-IK4 Research Alliance, Arrasate*
Spain (e-mail: aagirre@ikerlan.es)
**** Dept. Ingeniería Electrónica y Automática EPS de Jaén*
Spain (e-mail: eestevez@ujaen.es)

Abstract: Nowadays, it is more and more usual that applications provide critical and non-critical functionalities, the so-called mix-criticality applications. One of the main challenges for developing such applications is to isolate both application types, in such a way that non-critical functionalities do not interfere with the critical ones. The use of virtualization techniques like hypervisors can help to meet this objective. Indeed, a hypervisor is a software layer that provides hardware virtualization allowing different functionalities to be executed in different partitions which are temporally and spatially isolated. In this context, this paper proposes the use of modeling techniques to generate an initial set of constraints to the partition design necessary for a set of critical functionalities that coexist with non-critical ones. Furthermore, the proposed modeling approach supports all the development cycle of the component-based non-critical applications, from the design to the automatic generation of the skeleton code of their components.

1. INTRODUCTION

Current trends in distributed embedded systems (DES) point to a reduction in the number of physical network nodes in order to optimize costs and boost performance. The embedded computing is shifting to high performance hardware platforms (e.g. multi/many-core processors) that must provide support functionalities with different levels of criticality, such as safety-critical and non safety-critical functionalities (European Commission. Information Society and Media Directorate-General, 2012), (Pérez *et al.*, 2013). For example, many current cars are provided not only with safety-critical applications as the stability control, but also with other non-critical applications that provide added value functionalities. Both application types coexist and even share resources like the wheel sensor used by the stability controller and by the non-critical navigation system.

In addition, as applications with different level of criticality have different needs for example of timeliness and security, it is essential to avoid that non-critical applications interfere with the critical ones. Several challenges arise from that separation need. Particularly, technologies to guarantee the time and spatial isolation between critical and non-critical functionalities are needed. As it is detailed in (Burns and Davis, 2013) researches of mixed criticality systems date from 1987. After initial works dealing with a single processor, multiprocessor or multi-core platforms were introduced. In that sense, hypervisor technologies play a significant role, as they provide the logical partitioning capability of the underlying hardware through the virtualization of its physical resources (e.g. CPU, memory) (Crespo, Ripoll and Masmano, 2010).

Such partitioned systems require a configuration regarding the definition of the partitions and the allocation of the applications to those partitions. Therefore, for a good configuration design it is necessary to have a detailed characterization of the system itself as well as a deep knowledge of functional and non-functional requirements of the applications that will run on it. The use of models introduces the abstraction necessary to avoid non relevant aspects of a system, in such a way that each stakeholder just works with information related to its viewpoint (Brambilla, Cabot and Wimmer, 2012). As a consequence, application specification can be separated from system characterization which can help to automate, validate and guide the partitions design (Schmidt, 2006).

As it is presented in (European Commission. Information Society and Media Directorate-General, 2012), most works about mixed criticality systems are focused on mixed time and safety critical system, and they are related to the development of hypervisors, scheduling and analysis techniques, hardware virtualization, certification technologies, etc. In this context, models have been applied in some research projects such as (CERTAINTY, 2012) and (MultiPARTES, 2013). In the former one, modeling languages are extended in order to achieve a formal component-based design language and a design methodology for real-time mixed critical embedded systems on multi-core processors, but with the aim of improving the certification process. The latter one proposes model-driven development methodology and tools for building trusted embedded systems with mixed-criticality components on multi-core platforms (Alonso *et al.*, 2013), based on the XtratuM hypervisor (Crespo *et al.*, 2010).

Taking into account the existence of a hypervisor that assures the temporal and spatial isolation of the target applications (mixed-criticality systems where critical applications coexist with non-critical ones), this paper proposes a model based approach to generate the constraints to define the minimum required number of partitions from a set of applications that share resources of the infrastructure using a hypervisor. This partition design is proposed according to: (1) their need of system resources, (2) their restrictions on resource usage, and (3) criticality kind. For that purpose, the proposed approach takes into account the characterization of both system resources and applications. As their special needs demand, critical applications must be designed and developed following the standards and methodologies that assure their correctness. For example, in case of hard-real time applications they could be modeled using the MARTE UML profile (OMG, 2011) and analyzed by means of temporal analysis tools such as MAST (González, 2013). Therefore, although these applications are taken into account for defining partitioning requirements, their design and development are not directly related with this work. However, this modeling approach does support the design and development of component-based non-critical applications, allowing the automatic generation of the skeleton code of their components.

The remainder of this paper is as follows: Section 2 illustrates the proposed modeling approach that includes system resources and mixed-criticality applications. Section 3 details the design process to obtain an initial set of constraints to partition design. Section 4 presents a case study focused in a safety railway braking system where safety and non-safety functionalities coexist. Finally, Section 5 comprises some conclusions and future work.

2. MODELING APPROACH

This section applies modeling techniques to support the design and development of mixed-criticality systems that provide critical and non-critical functionalities, and that execute on platforms whose partitions are managed under a hypervisor's control similar to XtratuM. On the one hand, the proposed modeling approach allows the definition of the minimum requirements or constraints for the necessary partitions. On the other hand, all the development cycle of the non-critical and component-based applications is supported.

With this purpose it is necessary to characterize system resources and applications, including their needs of resources. Therefore, two domains have been identified for the modeling approach, which are detailed in the following sub-sections: (1) application specification and (2) resource characterization.

2.1 Application Specification View

This view contains the characterization of applications taking into account functional and non-functional requirements. The meta-model that contains the elements of this view and the relationships among them is presented in Fig. 1.

It is important to remark that a system can be considered as mixed-critical with respect to different criticality criteria, from now on the criticality kind such as timing, safety, or reliability requirements. More precisely, this work deals with the co-existence of critical (*CriticalApplication*) and non-critical (*Non_CriticalApplication*) applications being also the last ones composed of components that communicate among them.

Critical applications are modeled and analyzed externally. In our approach they are considered as a monolithic application component that represents the functionality of the whole application, being characterized by its criticality kind (*criticalityKind*), by its resource needs on a concrete node (*deployedIn*) such as the *budget*, deadline (*D*), period (*T*), and by the related operating system (*O.S.*). This implies that the whole critical application will be deployed in a unique partition. Additionally, it may also have associated the corresponding design model (*associatedModel*), for example based on MARTE if it has real-time criticality.

As far as non-critical applications are concerned, application functionality can be represented as a collection of components interconnected, each providing a concrete functionality. A component interacts with others through its input port and/or output port. A connector represents the data flow between two components, that is, the data sent by a component through its output port and received by a following one through its input port. In addition, this data exchange can be performed under a condition. It is important to remark that the meta-model illustrated in Fig.1 can be extended in order to introduce application execution logic and behavior driven reconfiguration as it is presented in (Armentia *et al.*, 2011). Finally, a component can be implemented in many ways, the so-called component implementations (*Comp.Impl.*). The functionality provided is the same, but each one presents different resources needs as they can be deployed in partitions of different nodes.

The proposed modeling approach also contains a set of properties that characterize other non-functional requirements of these non-critical applications, their components and their implementations.

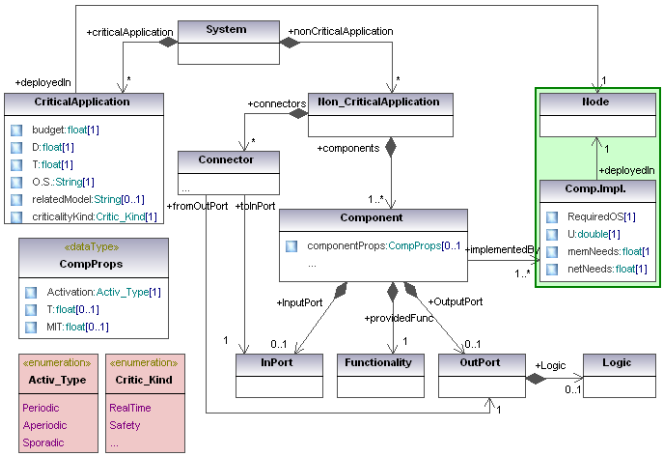

Fig. 1. Meta-model of the *Application Specification* view

Components can be characterized by a set of properties such as its activation type (*Activation*), deadline (*D*), and period (*T*) or minimum interarrival time (*MIT*) for periodic or sporadic components, respectively.

Component implementations and nodes are the nexus between both views. Therefore, the description of a component implementation must provide information about the needs of resources on a concrete node (*deployedIn*), essential to partitions design. With this purpose, memory needs, processor utilization (*U*), and network needs have to be recorded, as well as the operating system to run it.

2.2 Resources View

This view contains the description of the resources offered to applications. This work extends the characterization presented in (Calvo *et al.*, 2012), in order to cope with the new modeling requirements related to the critical aspects of a system. The characterization of this previous work, which was based on FRESCOR (González and Tellería, 2008) and MARTE UML profile, considers a node as a main resource that can be composed of a set of resources divided in five groups: (1) *memory*, to represent any storage resource; (2) *processor*, for describing every CPU; (3) *battery*, for power supply; (4) *operating systems*; and (5) *network interfaces* to connect to networks. This simplification considers that I/O-s can be modeled as peripherals (*Other* in Fig.2).

Fig. 2 presents the meta-model that corresponds to the extended characterization, with the changes highlighted in green. The concept of a node as a set of resources is maintained. More precisely, a node may consist of memory, processor, battery and network interface resources. Every processor can be supported by several operating systems, and it can be provided with more than one core. In addition, a node can be equipped with other peripheral devices, which can be shared or not. The hypervisor is in charge of the virtualization of this hardware, assuring time and spatial isolation among partitions.

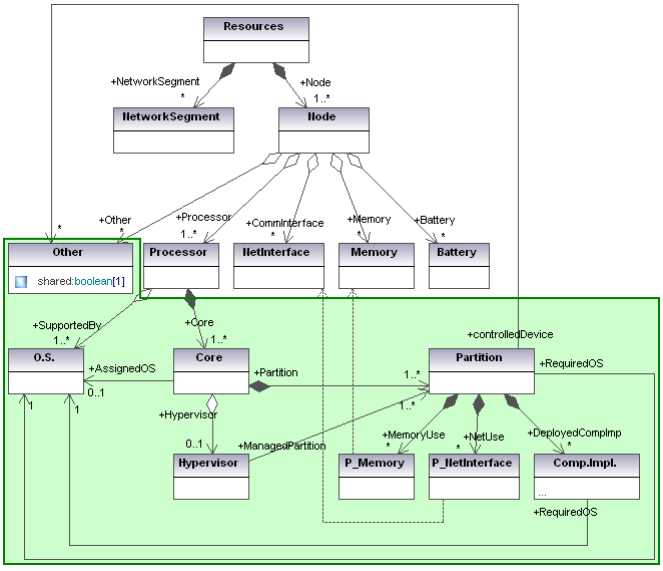

Fig. 2. Meta-model of the *Resources* view

Each partition also consists of a group of resources that correspond to their virtualization needs of basic hardware (memory and network, if the hypervisor supports it). Other peripheral resources are not virtualized, but they are managed by partitions. In case of a non-shared device its own partition is necessary. However, shared devices can be accessed through a unique partition and provided with an I/O server (Masmano, 2009).

Finally, component implementations are deployed in partitions, each one requiring an operating system to be executed. This implies that all the component implementations within a partition must require the same operating system. Note that component implementations cannot be deployed in partitions that manage peripheral devices.

3. DESIGN PROCESS: GETTING PARTITION REQUIREMENTS

This section describes the design process to follow in order to obtain a set of constraints for the minimum necessary partitions. It is important to remark that this is an initial configuration approach that has to be verified and completed. All the changes made by the application designer should be updated in the model.

3.1 Resources Characterization

Initially, the available system resources have to be characterized according to the meta-model proposed in Fig. 2, taking into account that, at this stage, partitions and component implementations are still undefined. Therefore, all the nodes and network segments within the system have to be described, identifying the available processor, memory, battery and network interface resources, as well as other peripheral devices. For example, a memory resource is characterized by its storage capacity and the throughput, i.e. the average memory access speed to memory.

3.2 Application Specification

The second step relies on the application specification. Firstly, critical applications must be characterized by means of their needs of resources in terms of budget, period and deadline. After, non-critical applications are modeled. Their functionality has to be defined as a set of components connected.

Every component's properties have to be indicated, and all the component implementations have to be characterized. It is necessary to indicate the node where the implementation will be deployed, as well as the required operating system (it has to be checked that it supports the processor of the target node), and its resources needs.

Finally, the skeleton code of the application components can be automatically generated by means of model-to-text transformations. The generated code contains the reception of data coming from previous components, the invocation to the

component's functionality, and the transmission of data to the following components.

3.3 Partition Requirements

The information captured in the models up to now is used to automatically propose an initial approach of the minimum partitions needed as well as their necessities and constraints.

Each non shared peripheral device might be managed by one and only one partition. In the same way, all the shared peripheral devices of a node might be managed by a partition.

For each critical application on a node, a dedicated partition is required.

Component implementations of non-critical applications deployed in the same node and that run on the same operating system are grouped in the same partition according.

4. CASE STUDY: SAFETY BRAKING SYSTEM

To validate the previous approach, a safety railway braking system based in three distributed voter notes plus a DMI (Driver-Machine Interface) node is proposed as an example of mixed-criticality distributed system. The braking system safety function must stop the train if certain conditions are met, whilst the non-safety part includes the graphical user interface to the system and the simulation of the environment (train sensors and actuators).

The safety functionality must ensure that the instantaneous speed of a virtual train does not exceed the limit established for the track section that is being crossed. There are three different limits with different actions to be taken. If the train speed exceeds the first limit the driver is warned through the DMI. If the second limit is crossed, the service brake is activated until the speed drops below the first limit, and finally, if the third limit is reached, the emergency brake is activated and the accelerator is deactivated until the train is completely stopped and the situation acknowledged by the train driver.

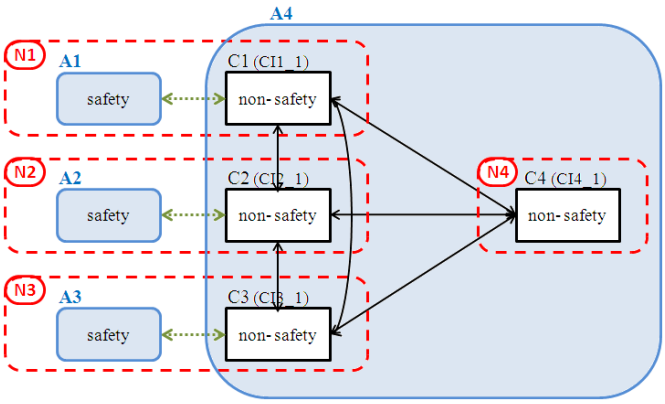

Fig. 3. Safety braking application

As far as resources are concerned (see Fig.3), the system is composed by three single core (with Hyper-Threading Technology) Intel® Atom™ based embedded nodes (N1, N2, and N3) plus a PC node (N4), that are distributed across an Ethernet network (black connections in Fig. 3). The application is partitioned and deployed over a TMR (Triple Modular Redundant) architecture.

From an application specification point of view (see Fig. 3), the system is composed of three critical applications with safety criticality kind (A1, A2, and A3), and a unique non-critical application based on four components (A4).

There are three safety applications that compute the same safety function in parallel and perform a 2oo3 (two out of three) voting of their results (outputs), based on odometry information computed from speed sensor and balise data inputs (A1, A2, and A3). They are diverse in terms of implementation technology (IEC, 2010), but perform the same safety function that eventually must decide to stop the train based on the 2oo3 voting mechanism. This way, a triple redundant architecture is achieved. Each safety application shares sensor and odometry data with the other two nodes (green dotted connections in Fig. 3). This data sharing is needed to perform a self-diagnosis based on a "reciprocal monitoring" pattern, and it is performed through the non-safety components (C1, C2, and C3).

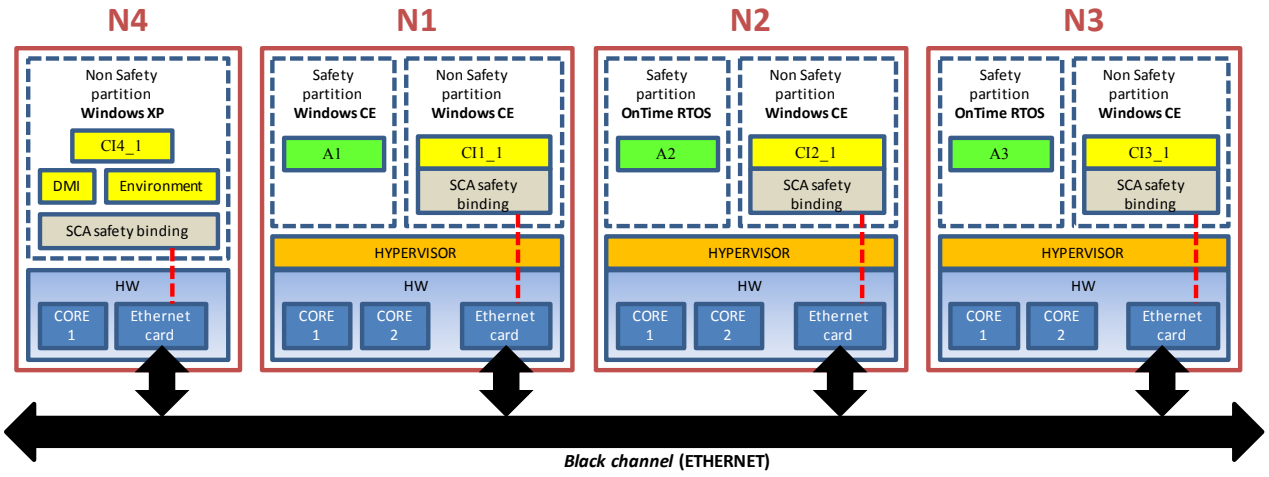

Fig. 4. Safety braking system architecture

As a result, the safety code is reduced considerably and so the certification costs. This can be done because the safety function is isolated from an eventual malfunction of the non certified components: If a non-safety related component fails, e.g. the communication related component, the system can detect it and force the system to a safe state (stopped). In general, failures that could jeopardize safety must be detected through diagnosis and managed in a way that a fail-safe state can be reached if necessary. Finally, non-safety components send data to another non-safety component to be shown on a DMI (C4).

Every component is implemented by a unique component implementation. For example, there are three component implementations CI1_1, CI2_1, and CI3_1 for the C1, C2, and C3 components, respectively. Fig. 3 also shows in red color the node where the implementation of each component and the critical applications will be deployed in.

The operating systems required by these component implementations and critical applications are as follows:

- Microsoft Windows CE, for the implementations of the components C1 (CI1_1), C2 (CI2_1), C3 (CI3_1), and for the safety application A1.
- Microsoft Windows XP, for the implementation of the component C4 (CI4_1).
- On Time RTOS, for the safety applications A2 and A3.

To obtain the constraints of the minimum partitions required, it is necessary to apply the rules described in the Section 3.3 to this characterization of resources and this application specification. As a result, one safety partition and a non-safety one are needed in nodes N1, N2 and N3. Fig. 4 presents a possible deployment diagram for the component implementations, which corresponds with this initial design.

The so-called safety partitions communicate with their corresponding non-safety ones through a shared memory mechanism (green dotted connections in Fig. 3) supported by the hypervisor, as it is more detailed in Fig. 5.

The safe communication over the Ethernet network among the component implementations CI1_1, CI2_1, and CI3_1 is performed by means of the SCA (Service Component Architecture (OASIS, 2011)) safety binding presented at (Agirre *et al.*, 2013). It provides the safety mechanisms described in the 61784-3-3 standard (IEC, 2007) to achieve safe communication over a "*black channel*", and at the same time abstracts the developer from low level communication details, as far as it provides a generic SCA binding implementation.

Finally, the skeleton code of the application components is automatically generated, by means of model to text transformations applied to the application model, according to the SCA component model (Armentia *et al.*, 2013).

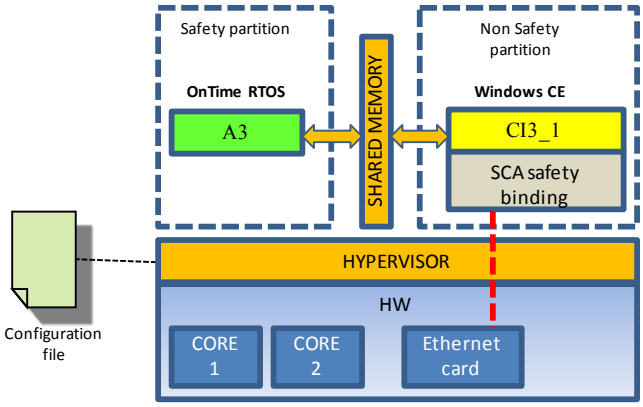

Fig. 5. Shared memory communication in N3 node

5. CONCLUSIONS AND FUTURE WORK

Advances in technology made possible to increment the processing capabilities of embedded systems, which resulted in the development of more and more complex applications, where critical and non-critical functionalities must coexist. Recent hardware virtualization techniques, such as the hypervisors, have allowed isolating the execution of several partitions, in a temporal and spatial way. Hypervisors require a good configuration design of the partitions, including the allocation of the corresponding applications.

This paper has proposed a modeling approach that, on the one hand, based on a characterization of the system resources and the specification of the applications that will run on it, provides the constraints to define the minimum required number of partitions. Note that target applications are critical applications that coexist with non-critical ones based on components; so the hypervisor configuration deals with the allocation of applications and components in partitions, regarding to the criticality kind of the application, the resources needs and restrictions on resource usage of their components.

On the other hand, this modeling approach supports the development cycle of the distributed non-critical applications based on components. By means of model-to-text transformations it is possible to generate the skeleton code of the composing components. Additionally, the modeling approach can be extended in order to capture all the properties relevant to the necessary future analysis, as for example scheduling analysis, as well as in order to define application and reconfiguration logic of non-critical applications.

As critical applications are fixed and must be certified it has been decided not to distribute them. However, as network certification processes are developed, future work can be aimed at applying this modeling approach to also distribute them.

ACKNOWLEDGEMENTS

This work was financed in part by the University of the Basque Country (UPV/EHU) under project UFI 11/28, by the Regional Government of the Basque Country under Project

IT719-13, by the MCYT&FEDER under project DPI 2012-37806-C02-01, and the European project FP7 MultiPARTES under project No.287702.

REFERENCES

Agirre, A. et al. (2013). SCA Extensions to Support Safety Critical Distributed Embedded Systems. In: 18*th* *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Cagliari, Italy, 10-13 September 2013.

Alonso, A. et al. (2013). Towards model-driven engineering for mixed-criticality systems: Multipartes approach. In *Proceedings of the Conference on Design, Automation and Test in Europe*. Grenoble, France, 18-22 March 2013. [Online] Available at: http://atcproyectos.ugr.es/wicert/downloads/wicert_papers/wicert2013_submission_6.pdf [Accessed: 4th March 2014].

Armentia, A. et al. (2011). Achieving reconfigurable service oriented applications using Model Driven Engineering. In: *16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Toulouse, France, 5-9 September 2011.

Armentia, A. et al. (2013). A MDE-Based Tool Suite for Dynamically Reconfigurable Service-Oriented Systems. In: *IV Congreso Español de Informática (CEDI)*. Madrid, Spain, 17-20 September 2013.

Brambilla, M., Cabot, J. and M. Wimmer (2012). Introduction. In: *Model-Driven Software Engineering in Practice*. Morgan & Claypool.

Burns, A. and R. Davis (2013). Mixed Criticality Systems - A Review. [Online] July 2013. Available from: http://www-users.cs.york.ac.uk/~burns/review.pdf. [Accessed: 4th March 52014].

Calvo, I. et al. (2012). Towards an infrastructure model for composing and reconfiguring cyber-physical systems. In *6th International Conf. Ubiquitous Computing and Ambient Intelligence (UCAMI)*. Vitoria-Gasteiz, Spain, 3-5 December 2012. **7656**, p. 282-289.

CERTAINTY project (2012). *CERTAINTY (Certification of Real Time Applications designed for mixed criticality)*. [Online] Available at: http://www.certainty-project.eu/ [Accessed: 4th March 2014].

Crespo, A., Ripoll, I., and M. Masmano (2010). Partitioned Embedded Architecture Based on Hypervisor: The XtratuM Approach. In: *European Dependable Computing Conference (EDCC)*. Valencia, Spain, 28-30 April 2010. p. 67–72.

European Commission. Information Society And Media Directorate-General. (2012). Mixed Criticality Systems. *Report from the Workshop on Mixed Criticality Systems*. [Online] Available at: http://cordis.europa.eu/fp7/ict/embedded-systems-engineering/documents/sra-mixed-criticality-systems.pdf. [Accessed: 4th March 2014].

González, M., and Tellería, M. (2008). *Deliverable D-AC2v2: Architecture and contract model for integrated resources II*. [Online] January 2008. Available at: http://www.frescor.org/index.php?page=publications [Accessed: 5th March 2014].

González Harbour, M. et al. (2013) Modeling distributed real-time systems with MAST 2, *Journal of Systems Architecture*. **59** (6). P. 331-340.

IEC (2010) IEC 61508: *Functional safety of electrical/electronic programmable electronic safety-related systems*. 2.0 Ed.

IEC (2007) IEC 61784-3-3: *Industrial communication networks - Profiles - Part 3-3: Functional safety fieldbuses - Additional specifications for CFP 3*. 2.0 Ed.

Masmano, M. et al. (2009) XtratuM: a Hypervisor for Safety Critical Embedded Systems. In *11th Real-Time Linux Workshop*. Dresden, Germany, 28-30 September 2009. pp. 263-272.

MultiPARTES project (2013). *MultiPARTES (Multi-cores Partitioning for Trusted Embedded Systems)*. [Online] Available at: http://www.multipartes.eu/ [Accessed: 4th March 2014].

OASIS (2011) SCA: *Service Component Architecture Assembly Model Specification Version 1.1*. [Online] Available at: http://www.oasis-open.org/committees/download.php/44090/sca-assembly-spec-v1.1-csprd04.zip [Accessed: 5th March 2014].

OMG. (2011). formal/2011-06-02: 2011. *UML Profile for MARTE: Modelling and Analysis of Real-Time Embedded Systems, Version 1.1*. OMG. [Online] Available at: http://www.omg.org/spec/MARTE/1.1/PDF/ [Accessed: 4th March 2014].

Pérez, J. et al. (2013) A safety concept for a wind power mixed-criticality embedded system based on multicore partitioning. In *1st International Workshop on Mixed Criticality Systems*. Vancouver, Canada, 3rd December 2013. p. 25-30. [Online] Available from: http://www-users.cs.york.ac.uk/~robdavis/wmc/paper15.pdf [Accessed: 4th March 2014].

Schmidt, D.C. (2006). Guest Editor's Introduction: Model-Driven Engineering. *Computer*, **39 (2)**, p. 25 – 31.