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

Mixed Criticality in Control Systems

Alfons Crespo * **Alejandro Alonso** ** **Marga Marcos** ***
Juan A. de la Puente ** **Patricia Balbastre** *

* *Universitat Politècnica de Valencia, Spain*
(e-mail: {alfons, patricia}@disca.upv.es).
** *Universidad Politécnica de Madrid, Spain*
(e-mail: {aalonso, jpuente}@dit.upm.es).
*** *Universidad del Pais Vasco, Spain*
(e-mail: marga.marcos@ehu.es)

Abstract: The complexity of industrial embedded systems is increasing continuously. Companies try to keep a leading position by offering additional functionalities and services. Systems are to be composed by multiple sensor, actuation and computation subsystems running in a coordinated way on a distributed platform. Due to the increment in processor power, it is possible to allocate a large number of functions in the same platform. This gives rise to mixed-criticality systems, when components with different criticality levels coexist in the same processor. This approach can lead to prohibitive certification costs. A better approach is to rely on partitioned systems, based on a hypervisor that isolates each of the virtual machines in the system. Components with different criticality levels are allocated to different partitions, in order to prevent interferences. The aim of this paper is to introduce mixed-criticality systems, to introduce the most challenging research topics, and to provide some background on the most promising techniques and research activities.

Keywords: Computer control systems, real-time systems, mixed-criticality systems.

1. INTRODUCTION

The complexity of industrial embedded systems is increasing continuously. Companies try to keep a leading position by offering additional functionalities and services. Embedded systems evolve to what has been named cyber-physical systems, which aims at merging the physical and virtual worlds. They are composed of multiple sensor, actuation and computation subsystems running on a distributed platform. These computational and physical components must be coordinated, distributed, and connected, as well as robust and responsive. Sensors capture information of the physical world, which is used for controlling and providing functions and services to users.

The automotive industry is an example of this trend. Current premium vehicles contain around 70–100 computers, around 100 electric motors and 2 km of wiring. These are connected by a mixture of databus standards (Thompson, 2012). Complex and assorted functions are currently included, such as driver assistance features, ESP systems, motor control, etc. It can be expected that more services will be added in the near future. Furthermore, drivers will receive a plethora of services by the connection of the vehicle to internet, like weather and traffic information, stations or food location, breakdown or accident assistance, etc. This panorama also holds for other application domains, such as aerospace, ubiquitous systems, manufacturing equipment, etc.

The large improvement on the execution and communication platforms is making this scenario feasible. Processors with an increasing number of cores can provide a great computational power to developers. The ubiquitous allowance of mobile networks with high bandwidth is commonplace. The most natural trend is to integrate a large number of functions in the same processor, in order to provide cost-effective systems. In this way is also possible to satisfy the requirements for reduced size, weight, and power consumption (SWaP).

Society relies on the devices that control cars, brakes, trains, or medical equipment. The envisaged future systems must continue to behave in a safe, secure and robust way. From this point of view, the integration of a large number of functionalities in the same execution platform poses a number of new technical challenges.

In some domains, it is necessary to certify 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 that the system will behave as expected, and is usually performed by an independent organization. Conformance to a safety standard is of great help, or even required, for certification. There are several such standards for different domains, such as electronic systems (IEC 61508), airborne civil avionics (DO-178B), nuclear power plants (IEC 880), medical systems (IEC 601-4), European railway (EN 50128]), European space (ECSS), etc.

Most of these standards rely on the assignment of integrity or criticality levels to the different components of the system. These levels represent the likelihood of a safety-related system for 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.

Traditionally, components with different criticality levels were located on separated processors, in order to prevent undesirable interferences. However, this approach often results in excessive resource consumption, and is being replaced by so-called mixed criticality systems that are being made possible by the increased processor power available nowadays. In *mixed-criticality systems*, components with different criticality levels coexist on the same execution platform. In this scenario, certification authorities would require the certification of the whole system, even the less critical parts, which would likely result in the cost of certification rising to prohibitive levels.

An alternative approach is to use virtualization in mixed-criticality systems. Under this approach, a hypervisor implements partitions or virtual machines that are isolated from each other in the temporal and spatial (i.e. storage) domains. Applications with different criticality levels can be located in different partitions, so that there are no undesirable interferences. In this way, only the hypervisor and the critical partitions have to be certified to the highest levels.

The aim of this paper is to provide a general view of mixed-criticality systems, their technical challenges, and some research results. Section 2 introduces mixed-criticality systems, the main research challenges related to them, and an overview of some relevant research projects. An analysis of partitioned systems is included in section 3. The scheduling of these systems for guaranteeing time requirements is the content of section 4. Mixed-criticality will require comprehensive and integrated toolsets, whose requirements are described on section 5. Finally, section 6 includes some sample applications illustrating future mixed-criticality systems.

2. MIXED CRITICALITY SYSTEMS

2.1 Overview

The following features will be present in a number of next generation embedded systems:

- High computer performance: a large number of the embedded applications will be multi-core, which provides high computing power to the users. As it has been a common trend, systems developers will try to take advantage of this feature for providing the most advance applications.
- Systems interconnection: the availability of networks of different types nearly everywhere has motivated the development of applications that interact and cooperate with external services. The development of distributed applications or ubiquitous systems is common nowadays.
- Requirements on size, weight and power (SWaP): Many embedded systems have this kind of requirements. A large number of them will be mobile, while others will be embedded on other equipments. Improvements on SWaP features will allow the development of devices that are easier to carry or with larger autonomy.
- Non-functional requirements: these type of requirements are not directly associated with a specific function or component of the system. They usually apply to the system as a whole. Non-functional requirements are usually defined as constraints on the system functionality. Time, reliability, availability, safety, or security, are examples of non-functional requirements.

  Time requirements are of specific interest in the development of control systems. The outcomes of the application have to be produced within a given time interval. Otherwise it is considered to be faulty. System developers have to ensure that time requirements are always met for safety critical applications.
- Functional complexity: processor performance makes it possible to encapsulate a large number of functions within one system, in order to produce competitive devices. This complexity poses a number of challenges to consider.
- Coexistence of applications with different safety and security levels: the requirement for integrating a number of applications implies that they will be of different nature. It is no longer advisable to isolate applications with more demanding requirements on a processor, due to the different type of associated costs. They must coexist in the same computer with other applications, while behaving as expected.

Mixed-criticality systems (MCS) are characterized by the integration of critical and non-critical applications on the same computing platform, as defined in the last item of the above list.

In order to fulfil these requirements a strong isolation of applications (critical and non-critical) is needed. An application is isolated from others if its execution is not influenced by the behaviour of the other applications. Different kinds of isolation can be considered:

- Fault isolation: a fault in an application must not propagate to other applications. Any fault must be handled either by the failing application itself or by the system.
- Spatial isolation: applications must execute in independent physical memory address spaces. The system must control that applications cannot access any memory areas that have not been specifically allocated to them.
- Temporal isolation: the real-time behaviour of an application must be correct independently of the execution of other applications. The allocation of the system resources to an application is not influenced by others, and can be analysed in a independent way.

In section 3, partitioned systems based on partitioning kernels are proposed as the most suitable software solution to fulfil these requirements. The partitioning kernel enables applications to be encapsulated in partitions that provide independent execution environments on a common execution platform. A partition is a container for a number of applications and an operating system supporting their execution. A virtualization layer or hypervisor is proposed in the same section as a convenient mechanism to implement the concept of a partitioning kernel.

The major benefit of using a partitioning approach for mixed-criticality systems is to reduce certification costs in

complex systems. In order to achieve this goal, it must be possible to analyse each application in an independent way. This requirement has some implications on the overall system:

- The hypervisor has to be certified at the same criticality level as the most critical application.
- System resources (CPU time, memory areas, IO ports, etc.) have to be allocated to partitions in a predefined and static way. Static allocation is necessary to enable independent analysis and certification of partitions. This principle applies both to the applications running in a partition and to the underlying operating system that controls their execution within the partition.
- Non-critical applications running in separate partitions do not have to be certified, as long as it can be guaranteed that they do not affect to the execution of critical partitions.
- Re-certification of a partition should not affect the certification status of other certified partitions.
- Incremental certification is a goal in order to achieve independent certification.

Static allocation of resources is a major requirement for achieving the previous described aims. From the execution point of view, the virtualization layer should execute partitions under a static allocation of temporal windows to the partitions. In this view, cyclic scheduling policies for partition execution seems the most appropriated approach, as proposed in the ARINC-653 standard (AEEC, 1996).

2.2 Research Challenges

There are several open issues in research with respect to the development of mixed-criticality embedded systems. A number of recent publications deal with the future challenges of mixed-criticality systems (Barhorst et al., 2009) (Thompson, 2012) (Burns and Davis, 2013). Some of them can be summarized as:

- *System modelling:* The development process should start with a description or model of the system under development. There is a need to define notations that allow providing all the functional components. In addition, it is required to find ways to describe other types of information relevant for system partitioning and deployment.
- *Methodology and development tools:* The development of mixed-criticality includes additional activities, such as partitioning or system integration, that are not common in previous systems. As an illustration, the development process involves a "system integrator" role, with responsibilities in the design of the system partitioning, assignment of resources to partitions, assignment of responsibilities to partition suppliers, and integration of the partitions. It is required a methodology and tools to guide this process and validate its outcomes.
- *Scheduling techniques for MCS:* Scheduling policies and scheduling techniques to achieve the independent certification of critical partitions have to be consolidated. The scheduling problem is one of the most active research area, resulting in proposals of different approaches to deal with partitioned systems. However, techniques that can guarantee the incremental scheduling of partitions are still needed.
- *Support for multi-core platforms:* Shared hardware resources in multicore systems have an impact on temporal isolation. The use of shared resources (such as L2/L3 cache, memory, bus, IO, etc.) by partitions running on different cores in parallel introduces an interference in the overall execution. This interference impacts directly in the independent behaviour of partitions and, as consequence, in its independent certification.

2.3 Research Projects

The great industrial interest of mixed-criticality systems has motivated the definition of a research roadmap and challenges, such as in (Thompson, 2012) and (Barhorst et al., 2009). It has been also a priority topic on European funded research projects. Some of the most relevant projects with research activities of direct interest for the development of mixed-criticality systems are now listed.

ACROSS (ARTEMIS CROSS-Domain Architecture (ACROSS, 2013)) This project aims at designing a cross-domain architecture for embedded Multi-Processor Systems-on-a-Chip (MPSoC) and implementing a first version in an FPGA. The ACROSS MPSoC will provide a stable set of core services as a foundation for the component-based development of embedded systems with short-time-to-market, low cost, and high dependability. In order to facilitate system development, a library of middleware services will be realized for offering basic services to be used in multiple application domains (e.g., fault-tolerance, diagnosis, and security) Another significant result of the project will be a general design methodology, supported by appropriate adaptable tools, for the implementation of ACROSS-based applications.

ARAMiS: (ARAMiS: Automotive, Railway and Avionics Multicore Systems (ARAMiS, 2013)) Its objective is to develop support for the appropriate deployment of multicore systemas and virtualization in the domains of automotive, avionics and railway, especially for safety related systems. The target systems will be multicore and relying on virtualization for providing safety critical applications in mobility domains. Some relevant topics in this project include time requirements, performance, reliability, availability, security and energy efficiency.

CERTAINTY: (CErtification of Real Time Applications desIgNed for mixed criticaliTY (CERTAINTY, 2013)) It addresses the certification process for mixed-critical embedded systems featuring functions dependent on information of varying confidence levels. The main challenge is to increase the complexity of the systems, where time and safety critical solutions becomes even more complex on multi-core platforms as time disturbances, uncertainties, and unreliability are emerging side effects that need to be efficiently handled. The project relies on mixed-criticality approaches, with strong segregation and high levels of certification. Application domains include avionic, automotive and automation, where real-time and safety-critical requirements are of primary importance.

IMA-SP (Integrated Modular Avionics for Space (IMA-SP, 2011)). The European Space Agency (ESA) launched in 2011 this project to study the applicability of IMA architectures to space applications. The aim of the project was to define the requirements for the use of temporal and spatial partitioning systems (TSP) in the space domain, using the specific hardware available. The project was integrated by industrial partners and researcher teams who contribute from different views to the approach.

MCC: (Mixed Criticality Embedded Systems on Many-Core Platforms (MCC, 2013)) This project aims at reconciling the conflicting requirements of partitioning for safety assurance and sharing for efficient resource usage. The consortium aims at developing for multi-core platforms, deriving verification procedures for mixed-criticality systems, explore the theoretical bounds of the developed schemes, and developing the necessary run-time controls to manage the sharing of communication media between the criticality levels.

MultiPARTES: (Multi-cores Partitioning for Trusted Embedded Systems (MultiPARTES, 2013) (Trujillo et al., 2013)) This project is aimed at developing tools and solutions for building trusted embedded systems with mixed criticality components on multicore platforms. The approach is based on developing an innovative open-source multicore-platform virtualization layer based on the XtratuM hypervisor. A software development methodology and its associated tools will be developed, in order to enable trusted real-time embedded systems to be built as partitioned applications, in a timely and cost-effective way.

parMERASA: (Multi-Core Execution of Parallelised Hard Real-Time Applications Supporting Analyzability (parMERASA, 2013)) The goal of this project is to make it easier to use multi-core processors in the development of real-time systems. Current multi-core architectures make extremely difficult to measure the worst case execution time of a activity. In addition, time analysis mainly covers sequential process execution. This project will provide technical innovations for dealing with aspects such as parallelization techniques for safety-critical applications, timing analysable parallel design patterns, operating system virtualization, and efficient synchronisation mechanisms, or timing analysable multi-core architecture with up to 64 cores.

RECOMP: (Reduced Certification Costs Using Trusted Multi-core Platforms (RECOMP, 2013)) This project aims at the development of mixed criticality systems on multi-core architectures, where safety applications meet relevant standards, without affecting the efficiency and design cost of the lower criticality parts. The project has developed a variety of mechanisms for virtualization and safe core-to-core communication for different safety levels with different hardware and development costs. Methods and tools were also developed to model the architectures and validate their safety properties, such that the hardware/software mechanisms can be used in industrial design processes.

vIrtical: (SW/HW extensions for heterogeneous multi-core platforms (vIrtical, 2013)) The goal of this project is to increase functionality, reliability and security of embedded devices at sustainable cost, and power consumption. The project relies on virtualization as the basis for this aim. In particular, it tries to provide to the virtualization concept in embedded devices, the same maturity level as in the general-purpose computing domain, in terms of flexibility and security. In order to elaborate this approach, this project will deliver software/hardware extensions at different layers of the design stack (hardware, operating system, hypervisor and applications) to increase flexibility, programmability, performance, QoS, reliability, security and power saving.

3. PARTITIONED SYSTEMS

Partitioned software architectures were defined to create trusted systems. They have evolved to fulfil security and avionics requirements, where predictability is extremely important. A partition is an execution environment integrated by an operating system and a set of applications. Partitions are executed on top of a hardware platform, possibly virtual, in an independent way.

In order to achieve this partition independence, a separation kernel was proposed in (Rushby, 1981) as a combination of hardware and software, for allowing multiple functions to be performed on a common set of physical resources without interference.

The MILS (Multiple Independent Levels of Security and Safety) initiative is a joint research effort between academia, industry, and government to develop and implement a high-assurance, real-time architecture for embedded systems. The technical foundation adopted for the so-called MILS architecture is a separation kernel. In addition, the ARINC-653 (AEEC, 1996) standard uses these principles to define a baseline operating environment for application software used within Integrated Modular Avionics (IMA), based on a partitioned architecture.

Integrated Modular Avionics (IMA) was the solution that allowed the aeronautic industry to integrate new functionalities, while maintaining the level of complexity and efficiency. Its main goal was to define an architecture that captures and handles faults at the different levels and permits the parallel application development. One of the main aspects to cover fault management is the temporal and spatial isolation of partitions. Currently, the ESA (European Space Agency) has initiated a program for the development of Time and Space Partitioning (TSP) solutions for space systems. Some of the potential benefits identified of TSP are (Windsor and Hjortnaes, 2009):

- Provide a clear model of the development process, by enforcing the integrator and developer roles.
- Facilitate the integrator activities providing a set of assistance tools.
- Provide a common target for developing applications.
- Increase the efficiency of the software validation and qualification process.

Virtualization techniques are the basis for building partitioned software architectures. Although virtualization has been used in mainframe systems since the 60's, the advances in the processing power of desktop processors in the middle of the 90's, opened the possibility to use it in the PC market. The embedded market is now ready

to take advantage of this promising technology. The main elements on this technology are:

Virtual machine (VM) is a software implementation of a machine (computer) that executes programs like a real machine.

Virtualization layer is the software layer that virtualizes the computer resources. It can be defined and used for dealing with application resources or system resources.

Hypervisor (also known as virtual machine monitor (VMM)) is a layer of software (or a combination of software/hardware) that allows running several independent operating systems in a single computer. Hypervisors are virtualization layers involving hardware (CPU, Memory, etc.) and software.

The purpose of the hypervisor is to virtualise efficiently the available resources. One of its required features is that it should introduce low overhead; therefore the performance of the applications executed on the virtualized system should be close to the same applications running on the native system.

The current state of the virtualizing technology is the result of the convergence of several technologies: operating system design, compilers, interpreters, hardware support, etc. This heterogeneous origin, jointly with its fast evolution, has caused some confusion on the terminology. The same term is used to refer to different ideas and the same concept is differently named depending on the engineer background.

3.1 Types of Hypervisors

Attending to the resources used by the hypervisor there are two classes of hypervisors, usually called type 1 and type 2. Type 1 hypervisors run directly on the native hardware (also named native or bare-metal hypervisors); the second type of hypervisors is executed on top of an operating system, which is called host. The operating systems that are executed in the virtual environments are called guest OS. Considering the virtual environment provided by the hypervisor, there are two types of hypervisors: full virtualization hypervisors, and para-virtualization hypervisors, which are described in the following paragraphs.

Full Virtualization

Full virtualization provides a complete re-creation of the hardware behavior of a native system to each of the guests. The objective is that the guest system should not be able to detect whether the system is real or virtual.

It is important to distinguish between full virtualization and emulation. With full virtualization, the guest system is very similar to the native system. In fact, most of the guest code is executed directly by the native processor, while only those instructions that deal with the processor or peripheral management are virtualized (those privileged instructions that can break the temporal or spatial isolation are called conflicting instructions). Emulation is a technique for simulating in software the behaviour of a computer. The main difference between a full virtualizer and an emulator is that while the first one is focused on achieving the maximum performance by reusing the native resources as much as possible, the emulator tries to precisely reproduce the behaviour of the emulated system.

A common technique used in full virtualizers is to use the processor facilities that are not employed by the operating systems. For example, the x86 architecture provides 4 privilege levels (rings) which where initially designed to help operating system programmers, but which are seldom used in current operating systems. Full virtualizers use the ring 1 to virtualize the supervisor mode of the processor (the guest OS is moved from ring 0 to ring 1). The virtualization layer has to catch unprivileged accesses to or from guest operating systems, and deal with them in a transparent way. A full virtualizer is able to run unmodified versions of the software.

Para-Virtualization

Para-virtualization is a virtualization technique where the conflicting instructions are explicitly replaced by functions provided by the hypervisor. In this case, the guest system has to be aware of the limitations of the virtual environment and use the hypervisor services.

The para-virtualization technique greatly improves the performance and simplifies the hypervisor. With this technique it is possible to develop very compact and efficient hypervisors. The only drawback is that the guest system must be modified or ported to the hypervisor API. If the hypervisor API is close to the native processor operation, then the porting may require the modification of only a small number of instructions. The hypervisor is still in charge of managing the hardware resources of the systems, and enforcing the spatial and temporal isolation of the guests.

Para-virtualization is the technique that best fits with the requirements of embedded systems: faster, simpler, and smaller. The customization (para-virtualization) of the guest operating system is feasible when the source code is available. In addition, this technique does not requires special processor features that may increase the cost of the product.

3.2 Virtualization and Real-Time

Bare-metal hypervisor technology is the most promising approach to achieve the best performance, which is of major importance for designing and implementing critical real-time systems. On the other hand, para-virtualization, jointly with dedicated devices, permits to reduce drastically the size of the virtualization layer code. In order to design a hypervisor for safety critical systems, the following design criteria has to be considered:

Strong spatial isolation : A hypervisor has to be executed in privileged processor mode, whereas partions are executed in user processor mode. Partitions are allocated in independent physical memory addresses. Partitions cannot access other partition memory areas. The basic concern of spatial isolation is to detect and avoid the possibility that a partition can access another partition memory. The hardware uses to provide some basic mechanisms to guard against violations of spatial isolation.

Strong temporal isolation : It refers to the system ability to execute several partitions guaranteeing i) the fulfilment of the timing constraints of the partition tasks, ii) that the execution of each partition does not depend on the temporal behaviour of other partitions. The enforcement of temporal isolation is achieved by designing a schedulable plan and guaranteeing that it is executed as specified.

Supervisor partitions : Some partitions can use *special* services provided by the hypervisor. These services include: partition management, access to system logs, etc.

Partition management : Partitions are executed in user mode, thus guaranteeing that they have not access to processor control registers. The hypervisor defines a set of services for allowing the supervisor partitions to start, reset, reboot and stop partitions.

Robust communication mechanisms : Partitions shall communicate with other partitions using the specific services provided by the hypervisor. Th basic mechanism provided to the partitions is port-based communication. The hypervisor implements the link (channel) between partitions.

Interrupt Model : The hypervisor provides an interrupt model to the partitions. Partitions can not interact with native traps. All the interrupts are detected and handled by the hypervisor, and propagated to the partitions, according to the given interrupt model.

Fault management model : Faults are detected and handled by the hypervisor. The detection of a fault can be the occurrence of a system trap or the occurrence of an event generated by the hypervisor code. The hypervisor code includes a set of assertions to verify the properties of the system. All hypervisor services have a set of pre and post conditions that assert the system properties. A Health Monitor module within the hypervisor implements the fault management model. Actions associated to the Health Monitor depend on the partition or the hypervisor fault handling mode.

Resource allocation : Fine grain hardware resource allocation is specified in the system configuration file. This configuration permits to assign system resources (CPU, memory, I/O registers, devices, memory, etc.) to the partitions.

The ARINC-653 (AEEC, 1996) standard specifies the baseline operating environment for application software used within Integrated Modular Avionics (IMA), which is based on a partitioned architecture. Although not explicitly stated in the standard, it was developed considering that the underlaying technology used to implement the partitions is the separation kernel. Although it is not an hypervisor standard, some parts of the APEX model of ARINC-653 can be close to the functionality provided by an hypervisor.

3.3 XtratuM Hypervisor

XtratuM (Masmano et al., 2009, 2010) is a type 1 hypervisor that uses para-virtualization. The para-virtualized operations are as close to the hardware as possible. Therefore porting an operating system is a simple task: replace some parts of the operating system HAL (Hardware Abstraction Layer) with the corresponding hypercalls.

XtratuM is being used as a TSP-based solution for building highly generic and reusable on-board payload software for space applications (Arberet et al., 2009). A TSP based architecture has been identified as the best solution to facilitate and secure reuse. It enables a major decoupling of the generic features that are being developed, validated, and maintained in mission-specific data processing (Arberet and Miro, 2008).

In a hypervisor, and in particular in XtratuM, a partition is a *virtual computer* rather than a group of strongly isolated processes. When multi-threading (or tasking) support is needed in a partition, then an operating system or a run-time support library has to provide it. In fact, it is possible to run a different operating system on each XtratuM partition.

XtratuM was designed to meet safety critical real-time requirements. Its most relevant features are:

- Bare hypervisor.
- Employs para-virtualization techniques.
- Designed for embedded systems: some devices can be directly managed by a designated partition.
- Strong temporal isolation: fixed cyclic scheduler.
- Strong spatial isolation: all partitions are executed in processor user mode, and do not share memory.
- Fine grain hardware resource allocation via a configuration file.
- Robust communication mechanisms (XtratuM sampling and queuing ports).
- Static resource allocation. A configuration file specifies the temporal and spatial allocation of resources to partitions.

XtratuM is available for different processors: x86, LEON2/3/4, ARM Cortex R4. XtratuM has been adapted to deal with heterogeneous multicore architectures, in the framework of the MultiPARTES (MultiPARTES, 2013) project.

4. SCHEDULING TECHNIQUES FOR PARTITIONED SYSTEMS

4.1 Hierarchical scheduling

Partitioned software architectures define two layers: a partitioning kernel in charge of allocating resources to partitions, including CPU time, and the partitions themselves, where applications are executed. Since partitions usually include different threads or tasks that are executed concurrently, the local operating system must in turn manage the allocation of resources to the partition tasks. Such a scheme leads, in a natural way, to the concept of hierarchical scheduling, where there is a global scheduler that allocates CPU time to partitions, and a local scheduler in each partition, that schedules the execution of tasks within the processor time available to the partition.

A number of different scheduling schemes have been proposed for both global and local scheduling. These include cyclic or time slicing frameworks, dynamic priority and fixed priority scheduling. As an example, ARINC 653 defines a cyclic scheduler at the global level and a fixed-priority scheduler at the local level (ARINC 653-1).

A key aspect of a scheduling method for real-time systems is the ability to analyse the schedulability of the system at design time, in order to guarantee the deadlines of real-time tasks. In recent years, considerable effort has been made in order to provide exact schedulability analysis for hierarchical systems. Specifically, for fixed-priority hierarchical systems (where fixed-priority scheduling policy is used in both global and local schedulers) the work of Davis and Burns (2005) and Balbastre et al. (2009) provides an exact worst-case response time analysis.

In order to achieve hierarchical scheduling, several strategies can be used:

- Server-based scheduling: The improvement in aperiodic servers, as well as a better understanding of the isolation properties of these mechanisms, refocused the application of such servers to what was called "bandwidth servers" or "resource reservation protocols". Servers enable real-time tasks to execute in a dynamic environment under a temporal protection mechanism, so that each server will never exceed a predefined bandwidth, independently of its actual workload requests. In this way, servers act as application containers, providing temporal isolation to applications. In this approach, a separate server is allocated to each application. Each server has an execution capacity and a replenishment period, enabling the overall processor capacity to be divided up among the different applications. The advantages of this technique include the great amount of research available. However, its main drawback is the difficulty to handle complex task models.
- Compositional scheduling: The basic idea of this approach is to extend the classical and widely used "divide and conquer" strategy to the temporal requirements. This technique has been widely accepted as a methodology for designing large complex systems through systematic abstraction and composition. The complexity of each component is hidden and abstracted through a clean and well-defined interface.
  This approach has the following main advantage:
  · Clean isolation of scheduling concerns between partition developers and system integrator. The partition developers do not have to provide details about its internal operation (task attributes), just the temporal abstract interface of the partition.
  On the other side, it has some drawbacks:
  · The abstract interface is an upper bound of the real needs of the partition, therefore there is a non-negligible utilisation penalty. The more partitions are in the system, the less processor utilisation can be granted.
  · Inter-partition resource sharing may be difficult to implement and to take into account in the schedulability analysis.
  · The algorithm is not optimal. Some feasible systems cannot be scheduled with this approach.
  · The restriction imposed on the periods of the partitions (they must be harmonic) is a limitation needed to produce "efficient" schedules (as well as to be able to use a deadline-monotonic policy at partition level). Otherwise, the resulting schedule may be far from optimal.
- Flat scheduling: In many cases, it is difficult to hide the internal task structure of the partitions because of the need to specify execution flows conducted by input/output operations that involve tasks in different partitions. A flat model approach considers all tasks, independently of the partition where they belong, as a global system. A single global scheduler can then be in charge of managing all the tasks, and a global schedulability analysis can be carried out. The last step is to adapt the solution back to the partitioned system by grouping (trying to put together) the tasks of each partition in order to reduce the number of partition context switches.
  This approach has the following advantages:
  · Dependencies between tasks of different partitions can be analysed and solved.
  · Mature theory support for this model.
  · The resulting schedule (or scheduling policy) can be very efficient. Depending on the task model, it may be possible to find the optimal solution.
  It has also the following drawbacks:
  · If an optimized solution is desired, a deep knowledge of the timing attributes of all the tasks is needed in order to carry out schedulability analysis.
  · There is no clean separation of concerns between partition developers and system integrator, or even among partition developers.
  · A change of an attribute of a task may require the whole schedule to be reworked.

4.2 Scheduling in multicore systems

The multiprocessor scheduling problem consists therefore in finding a feasible schedule for n tasks running on m processors. In the following we assume that $n \geq m$. Multiprocessor real-time scheduling is intrinsically a much more difficult problem than monoprocessor scheduling. The main reason is that few of the results obtained for monoprocessors can be directly applied to the multiprocessor case (Davis and Burns, 2011).

From the perspective of scheduling, multiprocessor systems can be classified into three categories.

- Heterogeneous multiprocessor systems: processors are different.
- Homogeneous multiprocessor systems: processors are identical.
- Uniform multiprocessor systems: the rate of the execution of a task depends only on the speed of the processor.

Research has been focused so far mainly on uniform and homogeneous multiprocessor scheduling.

Multiprocessor scheduling has to solve two problems: the allocation problem, that requires deciding on which processor a task should execute, and the local scheduling problem, that is, when a task should execute. Regarding allocation, there are scheduling algorithms that permit migration (at task or job level) and algorithms that do not permit it.

Scheduling algorithms where no migration is permitted are referred to as partitioned, whereas those where migration is permitted (either at task or at job level) are referred to as global.

Partitioned scheduling

Under this approach, each task is assigned to a single processor. This has the following advantages:

- Task overruns have only consequences in the same processor.
- There is no penalty in terms of migration cost.
- The implementation uses a separate run queue per processor, rather than a single global queue in the global approach. This reduces overheads due to queue management.

On the contrary, its main disadvantages are:

- Finding an optimal allocation of tasks to processors is a bin-packing problem, that is NP-hard in the strong sense.
- There are task sets that are only schedulable if migration is allowed.
- Partitioned scheduling algorithms are not work-conserving, as a processor may become idle, but cannot be used by ready tasks allocated to a different processor.

Still, partitioning is widely used by system designers.

Typical memory allocators such as First-Fit (FF), Worst-Fit (WF), and Best-Fit (BF) have been used to solve the problem of finding good sub-optimal static allocation of tasks to processors. These heuristics use the task period parameter as the key for allocation. Others, such as FFDU (First Fit Decreasing Utilization) (Oh and Son, 1995), use the task utilization as a key for choosing the next task to allocate. These allocators combined with classical scheduling algorithms (FPPS or EDF) gives rise to the most popular partitioned scheduling algorithms, such as RMFF, EDFBF, RMFFDU, etc. A comparison of these allocation schemes can be found in (Zapata and Mejia-Alvarez, 2003).

Global scheduling

Global scheduling cannot be used to reduce the multiprocessor scheduling problem to many monoprocessor scheduling problems, contrary to partitioned scheduling. The fact that tasks are allowed to migrate in the global approach gives rise to many unexpected effects and disadvantages, which complicate the design of scheduling and allocation algorithms for the global scheme. The most significant problems are:

- Migration of tasks to processors introduce a high overhead in the system.
- Migration increases the information flow between processors. This kind of communication may require the use of shared memory or communication channels.
- Predictability is much lower than that associated to the partitioned scheme.
- Some scheduling anomalies may occur, for example the Dhall effect: tasks sets with very small utilization may be unschedulable (Dhall and Liu, 1978).

On the contrary, the main advantages of this approach are:

- There are typically fewer context switches/preemptions. This is because the scheduler will only preempt a task when there are no idle processors.
- An advantage of the global scheme is its generality. Since tasks can migrate from one processor to another, the processor system "could be" better utilized.
- Global scheduling is more appropriate for open systems, as there is no need to run load balancing/task allocation algorithms when the set of tasks changes.

To classify global scheduling algorithms we use the concepts of fixed job priorities or fixed task priorities. In the former approach, the priority of a task can only change at job boundaries, while in the latter all jobs generated by the same task have identical priorities. Some well-known global scheduling methods are:

Fixed job priorities:

- The best-known scheduling algorithm for global multiprocessor scheduling is the so-called global Earliest Deadline First (EDF), where jobs are dispatched to any available processor according to a global priority scheme following EDF rules.
- Srinivasan and Baruah (2002) proposed the EDF-US[u] algorithm that gives the highest priority to tasks with utilization greater than a threshold u.

Fixed task priorities:

- Global-FP in which the global priority scheme is based on fixed priorities.
- Andersson (2008) proposed a "slack monotonic" algorithm, where priorities are ordered according to the slack of each task. This algorithm is known as SM-US.

Dynamic priorities:

- Pfair (Baruah et al., 1996) is a schedule generation algorithm that is applicable to periodic task sets with implicit deadlines. Pfair is based on the idea of fluid scheduling, where each task makes progress proportionate to its utilization (or weight in Pfair terminology). Pfair scheduling divides the timeline into equal length quanta or slots. At each time quantum t, the schedule allocates tasks to processors, such that the accumulated processor time allocated to each task τ_i will be either $\lceil t_{ui} \rceil$ or $\lfloor t_{ui} \rfloor$. A number of variants on the Pfair approach have been introduced (ERFair, PD, PD2, BF).
- Lee (1994) introduced the Earliest Deadline until Zero Laxity.

5. REQUIREMENTS ON THE DEVELOPMENT TOOLS

The complexity of the development of mixed-criticality embedded systems can only be overcome by the availability of adequate methodologies and tools. Due to the technical difficulties of achieving this aim, it is not feasible to develop a single tool that is completely satisfactory. On the contrary, the solution should rely on a set of complementary and composable tools that deal with different aspects of the problem. There is a ongoing effort by the research community on defining such tools, as it is the

case with the research projects listed in section 2. The set of requirements that these tools must fulfil can be summarized as follows:

Development of mixed-criticality systems The concept of criticality is central in the whole development process. The criticality level of each system component must be stated. This property has to be considered in all the functionalities provided by the tools: partitioning, validation, artefacts generation, or testing.

Support for non-functional requirements These are of great importance when dealing with embedded systems. They are not directly associated with an specific function or components of the system. They usually apply to the system as a whole, and defined as constraints on the system functionality. Time, safety, and security, are examples of non-functional requirements that will be present in most of the targeted systems. The toolset has to provide means for specifying them, and validating their fulfilment. This support can be provided at different development phases. The toolset can include tools for validating a certain outcome with respect to a non-functional property. The generators and transformers have to consider them, in order to ensure that their outcomes are compliant.

Support for partitioned systems System partitioning is a fundamental activity on the target type of systems. However, there is little support in similar development tools. This toolset should generate system partitioning that has to be compliant with the system models, non-functional requirements, and hardware resources availability.

Support for multi-core architectures The execution platform can be multi-core, as it is commonplace in current industrial systems. It should be supported modelling multi-core systems and to assign partitions to cores, according to the model defined by the hypervisor

System modelling The toolset has provide means for modelling the whole system, which includes the applications, platform, and any other information that the user has to provide. This is required for ensuring a consistent and coherent handling of all the information related with a system development. It is also required to support legacy applications. This is mandatory for integrating applications that have been developed with other approaches.

Support system deployment Deployment is the last step required before running the system. When dealing with partitioned embedded systems, this implies the generation of a bootable software image that includes the hypervisor, the partitions, and their operating system and applications. The tools shall support system deployment by generating mechanisms for the automatic building of the system. System deployment also requires the configuration of the hypervisor, which includes information on the systems partitions, resources associated to them, etc. This system description can be naturally generated by the toolset, if the developer provides the required data.

6. APPLICATION TO THE DEVELOPMENT OF CONTROL SYSTEMS

Cyber-physical control systems (CPCS) are those that making use of computational resources (or cyber capabilities) interact with the external world (physical phenomena), in order to achieve the desired behaviour, i.e., to control some physical phenomena. In a sense, most embedded systems can be considered as CPCS. A number of CPCS may include functionalities with different criticality levels.

Example of CPCS can be found in different application domains, going from transport applications (avionics, automotive or railway) to medical devices or process industry. In such type of systems, it is common to distinguish between safety and mission criticality types. The former refers to the parts of the system whose failure can impact human safety, and the latter refers to the parts of the CPCS that allows achieving the purpose of the system. There may be functions that are critical for mission success that have no impact on safety issues. For instance in avionics, a field posing the highest levels of safety and security challenges, engine control is safety critical, while navigation or communication are generally considered mission critical.

The demanding requirement of current complex CPCS applications with respect to Size, Weight and Power (SWaP) is pushing CPCS to be implemented in mixed criticality architectures. At the same time, technology advances in communication and Internet technology is leading to networked control systems and larger scale complex systems. In this sense, innovation in the transport industry (automotive, aerospace and railway) is being introduced adding driver, operator or passenger services and interaction. Examples can be found in advanced driver assistance, automatic cruise control, autonomous driving or smart collaborative interfaces. Combination of real life and virtual reality may convert cars, aircraft and trains into smart personalized comfortable spaces (Thompson, 2012).

The consumer market may also benefit from mixed criticality architectures, as temporal and spatial isolation are key to ensuring safe system operation. Embedded computing based on multi-core and many-core processors will allow reducing design complexity and SWaP while providing increased flexibility. Thus, initiatives such as smart energy or mobility will lead to interconnected complex systems offering mobile services (so-called systems of systems). Applications of this type are under development on domains such as automation, smart metering, medical monitoring, traffic control or smart grid.

On the other hand, multimedia services arising in the consumer market are placing a strong demand for the same services to be available in transport applications. New services via Internet are being introduced in the automotive sector, such as navigation systems and remote support and configuration, providing better and faster customer services.

However, where mixed-criticality architectures are making a great impact, it is in safety-related applications where SWaP and increased processing capabilities are very demanding requirements. Future avionics are provoking a

movement towards all-Integrated Modular Architecture (IMA), as a means to provide more processing power while decreasing SWaP (B.Triquet, 2012). Barhorst et al. (2009) analyses the impact of mixed criticality architectures in the field of avionics, mainly from the perspective of UAVs. G. Horvath (2012) presents the first step towards a partitioned software architecture for space robotics, highlighting the benefits throughout the entire development cycle from requirements and design to implementation, testing and operations. They also identify the changes in flight software design as well as the risks associated to them.

Looking at the automotive sector, migration of vehicle security features from mechanical solutions to CPCS is another field in which mixed criticality architectures seem to have direct application. Brewerton et al. (2012) present a safety architecture based on the multicore extensions of AutoSAR for steering column lock, including the verification and validation of the selected mechanisms. The work concludes with some of the advantages of the mixed criticality architecture:

- cost reduction through the reduced number of ECUs needed.
- reduction in integration (verification and validation) costs by OEM, as a set of safety function can be integrated in a single ECU,
- scalability, being possible to deploy safety function on a single core or spread them over multiple synchronized cores.
- simplified certification effort, reducing development costs.

In summary, current requirements in SWaP, distribution, and reconfiguration capabilities make mixed-criticality architectures a good option for many of the future CPCS applications. But this type of architectures also leads to research challenges and new design scenarios. Software and hardware providing temporal and spatial isolation are needed, as well as a re-definition of criticality levels, since functionalities may have variable criticality defined by their context or use if reconfiguration is triggered. Criticality level may pass from being one-dimensional to a multi-dimensional issue depending on the CPCS behaviour. Mixed criticality architecture opens the road to change from design and certification to design for certification, as pre-certified components (hardware and software) would exist that can be combined with mixed criticality functionality components to ensure CPCS meet their functional and non-functional requirements. These are only some of the features of the CPCS of the near future belonging to different application fields such as aerospace, automotive and rail transport, wind turbines, power grid, medical or factory automation. As a consequence, new challenges arises demanding research effort in modelling, analysis and design methods and tools able to ensure that CPCS composed by functionalities that can operate in different configurations satisfy competing safety, security or timeliness requirements in all cases. Reference architectures, methodologies and tools for reducing certification efforts, parallelization techniques and tools for mapping functionalities to cores, timing analysis methodologies and tools and virtualization techniques, including network virtualization, are some challenges to start with.

7. CONCLUSIONS

This paper has discussed the use of mixed-criticality systems. These type of systems are the result of the evolution of embedded systems, which is characterized by more complex functionality, more powerful processors, requirements on size, weight and power, and non-functional requirements. The most promising approach is to rely on hypervisors that provide temporal, spatial and fault isolation between partitions. In this way, components with different criticality level are assigned to different partition, preventing undesirable interferences.

The previous sections describe mixed-criticality systems, their main characteristics, most relevant challenges, and some of the on-going work. This paper has the goal of serving as an initial starting point for researchers interested on a topic with a great potential in the near future.

ACKNOWLEDGEMENTS

The work at Universidad Politécnica de Madrid and Universidad Politećnica de Valencia has been funded by the European Union 7th Framework Programme (FP7), project MultiPARTES (IST 287702), and by the Spanish National R&D&I program, project HI-PARTES (TIN2011- 28567-C03).

REFERENCES

ACROSS (2013). Artemis cross-domain architecture. URL `http://www.across-project.eu`.

AEEC (1996). *Avionics Application Software Standard Interface (ARINC-653)*. Airlines Electronic Eng. Committee.

Andersson, B. (2008). The utilization bound of uniprocessor preemptive slack-monotonic scheduling is 50 In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, 281–283. ACM, New York, NY, USA.

ARAMiS (2013). Automotive, railway and avionics muticore. URL `http://www.projekt-aramis.de`.

Arberet, P., Metge, J.J., Gras, O., and Crespo, A. (2009). TSP-based generic payload on-board software. In *DASIA 2009. DAta Systems In Aerospace.*

Arberet, P. and Miro, J. (2008). IMA for space : status and considerations. In *ERTS 2008. Embedded Real-Time Software.*

ARINC 653-1 (2006). *Avionics Application Software Standard Interface (ARINC-653). PART 1 – REQUIRED SERVICES*. Airlines Electronic Eng. Committee.

Balbastre, P., Ripoll, I., and Crespo, A. (2009). Exact response time analysis of hierarchical fixed-priority scheduling. In *Proceedings of 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications.*

Barhorst, J., Belote, T., Binns, P., Hoffman, J., Paunicka, J., Sarathy, P., Scoredos, J., Stanfill, P., Stuart, D., and Urzi, R. (2009). A research agenda for mixed-criticality systems. Technical report. URL `http://www.cse.wustl.edu/cdgill/CP-SWEEK09MCAR`.

Baruah, S., Cohen, N., Plaxton, C., and Varvel, D. (1996). Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6), 600–625.

Brewerton, S., Willey, N., Gandhi, S., and Rosenthal, T. (2012). Demonstration of automotive steering column lock using multicore autosar® operating system. Technical Report Paper 2012-01-0031, SAE Technical.

B.Triquet (2012). Mixed-criticality in avionics, memorandum. Technical Report Paper X42ME1201821, 19/1/12, Airbus Technical.

Burns, A. and Davis, R. (2013). Mixed criticality systems - a review. Technical report, Department of Computer Science, Unviersity of York.

CERTAINTY (2013). Certification of real time applications designed for mixed criticality. URL `http://www.certainty-project.eu`.

Davis, R.I. and Burns, A. (2005). Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of 26th IEEE International Real-Time Systems Symposium*, 389–398.

Davis, R. and Burns, A. (2011). A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, 43(4), 35:1 –35:44.

Dhall, S.K. and Liu, C.L. (1978). On a real-time scheduling problem. *Operations Research*, 26(1), 127–140.

G. Horvath, S. H. Chung, F.C.B. (2012). Safety critical software architecture: A partitioned software architecture for robotic spacecraft. Technical report, NASA Technical Report. URL `http://trs-new.jpl.nasa.gov/dspace/handle/2014/42075`.

IMA-SP (2011). IMA-SP Integrated Modular Avionics for Space. ESA project. Coordinator: Astrium SAS. Contract ESTEC 4000100764.

Lee, S.K. (1994). On-line multiprocessor scheduling algorithms for real-time tasks. In *TENCON '94. IEEE Region 10's Ninth Annual International Conference. Theme: Frontiers of Computer Technology. Proceedings of 1994*, 607–611 vol.2.

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

Masmano, M., Ripoll, I., Peiró, S., and Crespo, A. (2010). Xtratum for leon3: an open source hypervisor for high integrity systems. In *European Conference on Embedded Real Time Software and Systems. ERTS2 2010.* Toulouse (France).

MCC (2013). Mixed criticality embedded systems on many-core platforms. URL `http://www.cs.york.ac.uk/research/research-groups/rts/mcc/`.

MultiPARTES (2013). Multi-cores partitioning for trusted embedded systems. URL `http://www.multipartes.eu`.

Oh, Y. and Son, S.H. (1995). Fixed-priority scheduling of periodic tasks on multiprocessor systems. Technical report, Department of Computer Science, University of Virginia.

parMERASA (2013). Multi-core execution of parallelised hard real-time applications supporting analysability. URL `http://www.parmerasa.eu`.

RECOMP (2013). Reduced certification costs using trusted multi-core platforms. URL `http://atcproyectos.ugr.es/recomp/`.

Rushby, J. (1981). Design and verification of secure systems. volume 15, 12–21. Pacific Grove, California.

Srinivasan, A. and Baruah, S. (2002). Deadline-based scheduling of periodic task systems on multiprocessors. *Inf. Process. Lett.*, 84(2), 93–98.

Thompson, H. (2012). Mixed criticality systems. Technical report, EU, ICT.

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.

vIrtical (2013). Software/Hardware extensions for heterogenous multicore platforms. URL `http://www.virtical.eu`.

Windsor, J. and Hjortnaes, K. (2009). Time and space partitioning in spacecraft avionics. In *IEEE Conference on Space Mission Challenges for Information Technology.*

Zapata, O.U.P. and Mejia-Alvarez, P. (2003). Analysis of real-time multiprocessors scheduling algorithms. In *In the 24nd IEEE Real-Time Systems Symposium (RTSS'03).*