An industrial view on the common academic understanding of mixed-criticality systems

Alexandre Esper
Geoffrey Nelissen
Vincent Nélis
Eduardo Tovar

CISTER-TR-180810

2018/07

An industrial view on the common academic understanding of mixed-criticality systems

Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, Eduardo Tovar

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: alexandre.esper@altran.com, grrpn@isep.ipp.pt, nelis@isep.ipp.pt, emt@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

With the rapid evolution of commercial hardware platforms, in most application domains, the industry has shown a growing interest in integrating and running independently-developed applications of different 1ccriticalities 1d in the same multi-core platform, with the objective of improving the performance/cost ratio of the system. Such integrated systems are commonly referred to as mixed-criticality systems (MCS). Most of the MCS-related research published in the state-of-the-art cite the safety-related standards associated to each application domain (e.g. aeronautics, space, railway, automotive). However, those standards are not, in most cases, freely available, and do not always clearly and explicitly specify the requirements for mixed-criticality systems. This paper addresses the important challenge of presenting the relevant information available in some of the safety-related standards, such that the mixed-criticality concept is understood from an industrialist 19s perspective. In addition, the paper evaluates state-of-the-art mixed-criticality real-time scheduling models and algorithms against the safety-related standards.

Real-Time Syst (2018) 54:745–795
https://doi.org/10.1007/s11241-018-9308-9

An industrial view on the common academic understanding of mixed-criticality systems

Alexandre Esper[1] · Geoffrey Nelissen[2] · Vincent Nélis[2] · Eduardo Tovar[2]

Published online: 10 May 2018

Abstract With the rapid evolution of commercial hardware platforms, in most application domains, the industry has shown a growing interest in integrating and running independently-developed applications of different "criticalities" in the same multicore platform, with the objective of improving the performance/cost ratio of the system. Such integrated systems are commonly referred to as mixed-criticality systems (MCS). Most of the MCS-related research published in the state-of-the-art cite the safety-related standards associated to each application domain (e.g. aeronautics, space, railway, automotive). However, those standards are not, in most cases, freely available, and do not always clearly and explicitly specify the requirements for mixed-criticality systems. This paper addresses the important challenge of presenting the relevant information available in some of the safety-related standards, such that the mixed-criticality concept is understood from an industrialist's perspective. In addition, the paper evaluates state-of-the-art mixed-criticality real-time scheduling models and algorithms against the safety-related standards.

Keywords Mixed-criticality · Safety-related standards · Real-time scheduling

✉ Alexandre Esper
aresper@criticalsoftware.com

Geoffrey Nelissen
grrpn@isep.ipp.pt

Vincent Nélis
nelis@isep.ipp.pt

Eduardo Tovar
emt@isep.ipp.pt

1 Critical Software S.A., Porto, Portugal

2 CISTER/INESC-TEC, ISEP, Porto, Portugal

1 Introduction

In the last decade and in most application domains, the industry has shown a growing interest in developing methods and tools to implement, deploy, validate, and certify independently-developed applications of different "criticalities" in the same multi-core platform, with the evident objective of improving the performance/cost ratio of the system. Such integrated systems are commonly referred to as mixed-criticality systems (MCS). All over the world, the industrial interest in MCS has manifested itself in the form of important investments placed into R&D projects and academics since long time started to manifest their interest as well. The research community that focuses on the real-time scheduling theory has actively taken part in these efforts regarding MCS. Their base application model has quickly developed into a mixed-criticality (MC) task model that is today well-accepted and used in most research works on the subject. This model is based on the model proposed by Vestal (2007) and in this paper we will focus on that specific model and its related work. This new MC task model is in essence the result of combining the standard hard real-time requirements (studied by the real-time research community since the 70s) with the notion of "criticality" of execution. When transposed into the industrial world, the applications that correspond the best to that MC model and its combined requirements are those in which a part of the core functionality is delivered by *safety-critical* components.

The introduction of new constraints and requirements into the theoretical models has unexpectedly unveiled a brand new research landscape. Into this virgin research field some of the seminal results in scheduling theory had to be restated and revalidated, and an entire body of knowledge was to be rebuilt. The popularity of MCS immediately soared up in the real-time research community, which has been evidenced by the sudden emergence of tracks, sessions, and workshops that are now entirely dedicated to MCS in most of the flagship conferences on real-time systems.

Since its conception, the MC model has been gradually gaining sophistication by incorporating multiple levels of criticality or probabilistic WCET estimates to mention a couple of examples. Each transformation of the model has been motivated and justified as a mean to better cope with the requirements of MCS. However, most of the safety-related standards are not freely available, and still, the real-time community is confronted with the challenge of proposing solutions with certification potential to current industrial problems, i.e., solutions that are compliant with the safety-related industrial standards requirements and constraints. Considering this challenge, the main contributions of this paper are: (i) a practical survey and interpretation of some of the main safety-related industrial standards, thus complementing the academic MCS survey by Burns and Davis (2013); (ii) the identification of key requirements that must be taken into consideration when designing MCS solutions with certification potentials.

It is important to highlight that the objective of this work is not to judge all the theoretical contributions on MCS proposed so far, but simply to provide the minimum background material for those motivated to propose solutions to current industrial problems and challenges of the safety-critical industry. It is our hope that theoretical works can benefit from the contribution of this paper, in the sense that they can rethink and strengthen their assumptions toward more robust models with certification potential.

In our discussions, we refer to recommendations and requirements for the design of safety-critical applications using the three following standards: the IEC61508 [generic electrical and/or electronic and/or programmable electronic (E/E/PE)] (IEC61508 2010), the ISO26262 (automotive domain) (ISO26262 2011) and the DO-178C (aeronautics domain) (DO-178C 2011); those three standards being the most commonly cited in the real-time research literature on MCS.

An important aspect that must be highlighted with respect to the focus of this work is that it does not aim at covering or debating hardware configurations, e.g., multi-core versus single-core. Nevertheless, with the current trend of evolution of hardware platforms toward multi-cores, mainly due to the need of sharing resources to improve on SWaP (size, weight, and power), our work inevitably biases its attention towards those architectures. Note however that all the concepts discussed hereinafter are also applicable to single-core hardware platforms.

Organisation of the paper In order to provide the reader with sufficient background information, we start in Sect. 2 with an overview of the safety assessment process as required by the standards for the development of safety-critical systems. That section briefly explains how in practice the safety requirements are derived and how the development assurance levels (DALs) are assigned to the system safety functions. With this background, we introduce in Sect. 3 the concept of a mixed-criticality system (MCS for short). In Sect. 4, we summarize the most important architectural considerations and requirements from the three above mentioned safety-related industrial standards, with respect to the development of MCS. We conclude the "industry-oriented" part of the paper by presenting in Sect. 5 the industrial solutions that prevail in the aeronautic and automotive application domains to design mixed-criticality systems in accordance with their domain-specific requirements. We then move to the "academy-oriented" part of the paper. We start in Sect. 6 by discussing the Vestal model, the theoretical model of MCS most commonly found in the academic literature. We focus on Vestals model because of its wide acceptance in the real-time system academic community to model MCS. We use Sect. 6 to highlight key aspects of that model that are not fully compliant with the safety-related industrial standards. As a preliminary step towards setting a model that is compliant with the safety-related standards, we list in Sect. 7 some key requirements that *must* be captured and taken into consideration in the model. Then in Sect. 8 we discuss some of the academic solutions that may be compliant with those requirements. Finally, the paper is concluded in Sect. 9.

2 System design and development assurance process

In order to understand the concept of MCS from the industrial perspective, it is necessary to firstly understand that the safety-related industrial standards do not explicitly specify requirements for MCS, but they do specify stringent requirements that must be met to ensure the *safety* of the system. Secondly, it is necessary to understand that "criticality" is a generic term commonly adopted to designate the type of systems that perform safety critical functions, i.e., functions that in case of failure can potentially lead to e.g., injuries, death or damage to properties or to the environment. In the safety-related standards, different types of more specific terminologies are use to designate those functions, as well as their different levels of “criticalities”, which depend on the severity of the consequences and the probability of occurrence of their failures. The generic concept of “criticality” is therefore used to determine the level of rigour required to develop those safety critical functions in such a way that the risk of failure can be brought to an acceptable level. Therefore, to avoid generalizations of terminology that may potentially lead to misinterpretation of the safety-related standards requirements, it is of uttermost importance to precisely understand the industrial process for assigning those “criticality levels” to those functions, and why these functions must be isolated from each other to be able to safely coexist in the same platform (i.e., an essential requirement for implementing an MCS). To achieve that goal, we summarize in this section, as an example, the system design and development process adopted in the aeronautic domain for assigning those “criticality levels”, more specifically known as development assurance levels (DALs).[1]

During a typical development life-cycle of a safety-critical system, the behavior and characteristics that are expected from the system are expressed in the form of a list of requirements. Those are developed based not only on the system operational requirements (what the system is expected to do), but also considering non-functional properties related to safety, security and performance, including timing and energy constraints. In order to ensure the safety properties of a safety-critical system, a *system safety assessment process* must be carried out as part of the development life-cycle to determine and categorize the failure conditions of the system (e.g. through a hazard analysis). As a result of the system safety assessment process, safety-related requirements are derived, which may include functional, integrity, dependability requirements and design constraints. These requirements are then allocated to hardware and software components, thereby specifying the mechanisms required to prevent a fault occurring or to mitigate their effects and avoid the propagation of failures.

To help understand the safety-critical system development life-cycle, we provide below an overview of the safety assessment process commonly defined by the standards, e.g., ARP4761 (1996) and ARP4754 (ARP4761A 1996). We hence briefly explain how, in practice, the *development assurance levels* (DALs) are assigned to the system safety functions. Note however, that even though the fundamental concepts are in essence the same across most safety-related standards, the approaches adopted in each case varies, which makes the description of a general cross-domain safety assessment process not a straightforward task.

2.1 General safety assessment process

The safety assessment process starts *at the system level* with a hazard analysis. This technique identifies the system hazards and assesses their severity (according to the domain specific severity scale) by taking into account the operational environment. After that, a fault analysis is performed to identify the failure conditions that can trigger

[1] Note that in Sect. 2.2 other terminologies used in other domains will be presented.

the identified hazards. The most known fault-analysis techniques that are used across domains are the Fault Tree Analysis (FTA) (see Sect. 4.1 of ARP4761 (1996)) and the Failure Modes and Effects Analysis (FMEA)(see Sect. 4.2 of ARP4761 (1996)).

FTA is a top-down analysis technique that proceeds down through successively more detailed lower levels of the design. It facilitates the safety assessment process in the sense that it identifies only the failure events that combined (or individually) can lead to the occurrence of the undesired top level event. If the failure rates of the basic events are known, the probability of occurrence of the top level event can be quantified.

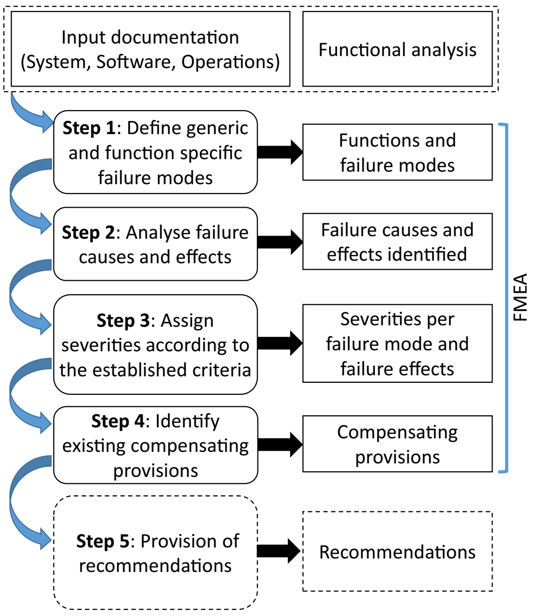
The FMEA on the other hand is a systematic bottom-up technique used to identify the failure modes of a system, function or component, and to determine their effects to the system upper levels. Contrary to the FTA, the FMEA addresses only failure effects that result from single failures.

FTA and FMEA are complementary techniques. Section 3.2 of ARP4761 provides a good overview of how these two techniques relate to each other and to the hazard analysis. After the identification of the failure conditions in the hazard analysis, the FTA can be applied to determine what single failures or combinations of failures can exist (or not) at the lower system levels that might cause each failure condition. The fault tree can then be complemented by the FMEA to ensure that all significant effects are identified as basic events (lowest level) of the tree. The FTA basic events can also get their failure rates from the FMEA. A lower level system item (e.g., component) contributes to the failure condition if it appears in one of the tree branches leading to the failure condition. A DAL is then assigned to the item according to the severity level of the worst failure condition that it contributes to.

Safety assessment also plays an important role at the software level, especially in the context of MCS, where the isolation of applications of different criticalities requires special attention. The inclusion of software errors in a qualitative manner in the safety analysis shows their contribution to the various failure conditions and can provide valuable information on deriving the DAL. The software safety assessment can also identify the specific safety related requirements for software such as containment boundary definitions, partitioning strategies, and specific verification strategies. Considering this, the criticalities of the applications need to be demonstrated by means of a fault analysis (either FTA, FMEA or both), which allows to assess the contribution of their software failures to the identified system hazards. However, when applying fault analysis techniques to software, the relationship between the top level hazards and software hazards or feared events (software specific anomalous behaviours) must be clearly determined. It is important to explicitly identify how a failure of a certain software function can affect other software and system functions, and whether they can potentially contribute or not to a system hazard.

The software hazard analysis is a top-down technique that makes recommendations to eliminate or control software hazards and relates the hazards to the interfaces between the software and the system. Software hazard analysis should ensure that the software does not interfere with the objectives and correct operation of the system. In the case where interference cannot be totally avoided the software hazard analysis must also evaluate and make recommendations to mitigate how the software can hinder the objective or operation of the system.

Fig. 1 Generic FMEA process


Input documentation (System, Software, Operations)
Functional analysis
Step 1: Define generic and function specific failure modes
Functions and failure modes
Step 2: Analyse failure causes and effects
Failure causes and effects identified
Step 3: Assign severities according to the established criteria
Severities per failure mode and failure effects
Step 4: Identify existing compensating provisions
Compensating provisions
Step 5: Provision of recommendations
Recommendations
FMEA

As explained in Sects. 4.1.2 and 4.2 of ARP4761, both FTA and FMEA can be used to perform a qualitative analysis of complex systems such as MCS, hence supporting the definition of mitigation measures, which may include hardware or software mechanisms and/or the inclusion of specific verification activities in the development process. We briefly describe the software fault analysis process using FMEA as an example. Figure 1 summarises the generic software FMEA process. The first step consists in performing a functional analysis, i.e., a listing and description of the software functions (rather than the items used in their implementation), and to define the generic failure modes to be applied to each function of each software component. The generic failure modes typically include incorrect execution, non-execution, or late execution of a software function. Therefore, based on the input documentation and on the previously identified software functions derived from the software requirements, the generic failure modes are mapped to the functions of each software component and the component's specific failure modes are then derived. Step 2 consists in analysing the component possible failure causes that can trigger the identified failure modes and on identifying the end effects of the failure mode to the system (possible failure propagation). During Step 3 severity categories (i.e., how bad is the consequence for the system) are assigned to every single failure mode. The severity of the consequences of the failure modes assigned in accordance with the severity assignment criteria defined specifically for the system being developed. Step 4 is the identification of existing compensating provisions that can either

1. circumvent or mitigate the effect of the failure,
2. control or deactivate product items to halt the generation/propagation of failure effects, or

3. activate backup or standby components to (at least partially) recover from the failure.

Generally speaking, design compensating provisions include:

– Redundant components or alternative modes of operation that allow continued and safe operation;
– Safety or relief feature (hardware or software) that allows effective operation or limit the failure effects.

Once the severity categories have been assigned to the failure modes and consequently to the respective software components, recommendations can be provided with the objective of reducing the risk associated with the potential critical faults identified (e.g. by increasing the amount and rigour of verification and validation activities).

2.2 Development assurance level

The software DAL establishes the necessary rigour of the development and of the verification and validation (V&V) activities that need to be performed on the software, in accordance with the adopted standard. The higher the DAL of a software, the higher the number of assurance activities that need to be performed, thus also increasing considerably the costs of its development. The process for the development of safety-critical software, which assures the software safety and dependability properties at a certain DAL is defined in several standards across several application domains (for instance, see Sect. 5.6 of ARP4761A (1996)). Typically, the DALs are divided into 4 or 5 levels, related to the categories of severity of a failure adopted by the standard. For instance, under the DO-178C (DO-178C 2011) standard (in the avionic domain), five severity categories are defined. These five categories are:

1. **Catastrophic** failures that result in multiple fatalities or the loss of the airplane.
2. **Hazardous** failures that result in serious or fatal injury to a relatively small number of occupants.
3. **Major** failures that reduce the capability of the airplane or the ability of the crew to cope with adverse operating conditions.
4. **Minor** failures that do not significantly reduce the airplane safety.
5. **No safety effect** failures that have no effect on safety.

The software DALs are then assigned depending on the severity category assigned to the failure(s) that may be caused by the analysed software component. Specifically, there are five levels defined as level A/B/C/D/E which are respectively assigned to software components (or modules) whose anomalous behaviour would lead to a system failure of catastrophic, hazardous, major, minor or negligible consequences. That is, a software that can potentially contribute to a catastrophic system failure shall be developed according to DAL-A requirements.

So far we have used DO-178C as reference for explaining the concept of development assurance level (DAL). However, IEC61508 and ISO26262 use different terminologies for describing the development process of a safety-critical or safety-related system, although the fundamental concepts are in essence the same.

IEC61508 defines the concept of system *safety function*. System safety functions are implemented by a safety-related system whose purpose is to achieve or maintain a safe state for the equipment under control (e.g. car engine) when a specific hazardous event occur. Associated to the system safety functions, the concept of *safety integrity* is defined, which refers to the probability of a safety-related system to satisfactorily perform the required safety functions under all the state conditions within a specified period. There are four *safety integrity levels* (SIL). The higher the safety integrity level of the safety function, the lower the probability that the safety-related system that executes that function will fail. Software SILs are used as the basis for specifying the safety integrity requirements of the system safety functions implemented by safety-related software. Although the SIL is composed of four levels, the IEC61508 does not explicitly define the failure severity categories and their association with the SIL. Only examples are provided that are not fully detailed. For instance, in Table C.1 of IEC61508-5 (IEC61508 2010), the following failure severity category levels are provided: catastrophic, critical, marginal and negligible. It is up to the system designers to define and detail those categories but it is important to note that the definition of those is based only on *qualitative* rather than quantitative measures. This note will be further discussed in Sect. 8.6.

ISO26262 derives from the generic IEC61508 and addresses the specificities of the automotive sector. ISO26262 defines the *automotive safety integrity level* (ASIL). Similarly to the SIL defined in IEC61508, the ASIL are composed of four levels, where D represents the most stringent and A represents the least stringent level in terms of requirements and safety measures (note that this is the exact opposite to the scale used by DO-178C). The higher the ASIL, the greater the needs to reduce the risk. Table 1 presents the risk matrix for the ASIL determination of hazardous events of automotive systems. It uses three parameters: "severity", "probability of exposure" and "controllability".[2,3] The severity defines the estimation of the extent of harm to one or more individuals that can occur in a potentially hazardous situation, the associated probability is the likelihood of the occurrence of harm, and the controllability is the ability to avoid a specified harm or damage through the timely reactions of the agents involved (e.g. the driver of the vehicle) possibly with support from external measures. Therefore, the ASILs explicitly consider one more parameter in comparison to the SILs, which is the ability to control failure effects. Note that, as it can be seen in Table 1, a high controllability (class C1) can often help reduce the ASIL of the components by 2 levels in comparison to the case where the controllability is almost inexistent (class C3).

2.3 Distinction between safety-critical, mission-critical and non-critical systems

Based on the safety assessment process presented in the previous subsections for assigning the DAL or the SIL, it is important to make the distinction between safety-

[2] A detailed description of these 3 parameters are outside the scope of this work. Please refer to ISO26262 (2011) for further details.

[3] In addition to the four ASILs, the class QM (quality management) denotes no requirement to comply with ISO26262 other than the project quality assurance requirements.

Table 1 ASIL determination for hazardous events (ISO26262 2011)

Severity class	Probability class	Controllability class		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

S severity class, *E* exposure, *C* controllability

critical, mission-critical and non-critical systems. To grasp the difference between those, it is fundamental to understand the difference between the concepts of safety and dependability (reliability, availability and maintainability). A safety-critical system can be understood as a system that, in case of severe failures, can lead to, e.g., injuries or death of individuals or group of individuals, or severe damage to environment, in accordance to the safety assessment performed according to the project criteria, as explained in the previous subsections. Examples of those types of systems are: aeronautic, railway and automotive systems. On the other hand, there are certain types of systems where the safety assessment process can demonstrate that even in a worst case failure scenario, the worst consequence that can potentially occur is the loss of the system, without any threat to human integrity or the environment. Mission-critical systems are commonly developed, for instance, for the space industry (e.g., satellite systems). In this case, it will be the organization or business that owns the system that will suffer the consequences of the failure, such as considerable financial losses. Therefore, the reliability, availability and maintainability (RAM) properties become the key concern when designing mission-critical systems. As an example, the reader is invited to consult Table 6-1 of standard (ECSS-Q-ST-40C 2009), which defines how the severity of potential consequences of undesirable events shall be categorized, both in terms of safety and/or dependability. As explained in clause 6.4.1b of ECSS-Q-ST-40C (2009), an understanding of the criteria defined in Table 6-1 (ECSS-Q-ST-40C 2009) shall be agreed between customer and supplier. For example, satellites developed for the European Space Agency (ESA) are typically developed to criticality level B (similar to the aeronautic DAL-B), as defined in Table D-1 of ECSS-Q-ST-80C (2009). Finally, non-critical systems are those types of systems where, according to the project criteria, even in a worst case failure scenario, the consequences will be minor, thus not significantly impacting the nominal system operation.

3 The notion of mixed-criticality systems

Considering the process presented in the previous section for assigning the DALs, the concept of mixed-criticality becomes straightforward to understand. A mixed-criticality system consists of applications of different DALs coexisting in the same system, sharing the same resources (potentially including the CPUs) but still preserving the safety characteristics of each individual application as required by the domain-specific safety-related standards. In addition, a MCS is first of all a critical system for which we want *simple* and *modular* designs. All safety-related standards in every application domain advocate the use of these principles in the design of such systems. These guidelines can be justified in a thousand ways, but in short, a simpler design guarantees a simpler conception phase and a simpler and thus less costly V&V, while modularity allows for better maintainability and easier upgradability.

The high-level process explained in the previous section for assessing the criticality of software systems is an important activity in the design of safety-critical systems, and consequently of mixed-critical systems. Based on the understanding of this process, one can conclude that there are two main solutions to reduce the criticality of a system component (i.e., to reduce the risk of severe failures):

1. Avoiding the propagation of faults between different components and in particular from low criticality components to higher criticality components;
2. Providing compensating provisions by adding effective mechanisms that could either prevent or mitigate the effects of a failure.

It also becomes clear that improving the reliability of the software, by reducing the risk of failures, is an essential step in the design of MCS.

Another important observation, considering the development assurance requirements derived from the risk assessment process described in the previous section, is that a certain system does not change its criticality during operation. Therefore, a mismatch in the interpretation of the concept of "system criticality" exists between the industrial standards and the academic papers, and is further discussed in Sect. 6.2. What happens in practice is that in the occurrence of a fault, the system may enter a different mode of operation, where less important functions can be dropped in benefit of the more important ones. This is related to the concept of graceful degradation, which will be discussed later in Sects. 4.2.4 and 6.5.

4 Requirements of safety-related industrial standards applicable to the development of MCS

After the introduction of the basic concepts, methodologies, and terminology for understanding and discussing MCS in Sects. 2 and 3, we summarize next key architectural considerations and requirements extracted from three safety-related industrial standards (IEC61508, DO-178C and ISO26262) that are deemed as most relevant for the design of MCS solutions with certification potential. We highlight the importance of this section since most of the safety-related standards are not freely available for the real-time community and even for industrialists it is very difficult to get access to a broad set of standards of several different domains.

4.1 The DO-178C

The DO-178C standard describes a set of important techniques that can be applied during the design of avionics systems, which may prevent software failures and/or limit or circumvent their effects on the system functions. To achieve that goal, the system safety assessment process needs to demonstrate that the software components will execute with sufficient independence. This independence must be ensured at the functional level, i.e., during the specification of the high-level software requirements, and at the design level, e.g., definition of common design elements, languages, and tools. If sufficient independence between software components cannot be demonstrated (e.g., through partitioning), then those components will be viewed as a single software component when assigning the software DAL. This implies that the DAL assigned to the components will be the DAL associated with the highest failure severity category that those components can contribute to.

Under DO-178C, the following safety-related software design methods are discussed: partitioning, dissimilarity (or redundancy), and safety monitoring. Dissimilarity is a design technique also referred to as multi-version software, where two or more different software components that perform the same functions are developed independently (Sect. 2.4.2 of DO-178C). It intends to avoid common sources of errors to contaminate the different versions of the same component. However, in the industry this technique is rarely applied due to its typically excessive cost and is thus not further discussed in this paper. Partitioning and safety monitoring as described in DO-178C are discussed in details below.

4.1.1 Partitioning

Similarly to IEC61508, DO-178C presents partitioning as one of the most important design instruments to safety-critical systems. The decision regarding the partitioning approach to be applied to a project must be taken during early phases of the software development life-cycle (Sect. 2.4.1 of DO-178C) and must address the following aspects: (i) the extent and scope of interactions that will be allowed between the partitioned components, (ii) how to isolate the components from each other, i.e., which protection strategy will be adopted (e.g through hardware functions or a combination of hardware and software).

Regardless of the adopted approach, DO-178C establishes five requirements for ensuring partitioning between the partitioned software components. The first requirement states that the code, input/output, or data storage areas of a software component cannot be contaminated by another software component that belongs to a different partition. The second requirement refers to the consumption of shared CPU time. A partitioned software component is only allowed to consume CPU time during its scheduled period of execution. The third requirement is related to hardware failures within a partition. Each partition should be able to contain the fault, i.e., a fault should not be able to propagate to the other partitions and hence cause failure of software components in those other partitions. The fourth requirement discusses the DAL level of the software application that provides the partitioning functionality to the system. This requirement states that the software that implements the partitioning functionality should have the same or higher DAL than the highest DAL of the software components assigned to any of the provided partitions. If the partitioning functionality is provided via hardware, the fifth requirement requires that a safety assessment must be performed on that hardware to ensure that in case of failure it will not cause failures on the software partitions and consequently affect the system safety.

4.1.2 Safety monitoring

As already mentioned in Sect. 4.2.2, safety monitoring (Sect. 2.4.3 of DO-178C) is a technique that allows the protection against specific failures through the generation of events (e.g., alarms) and activation of protective mechanisms when the monitored system function enters a faulty state. The safety monitoring functions can be implemented by hardware, software, or a combination of both. From the safety point of view, the safety monitor implements a safety barrier that will inhibit the failure of a software component from propagating throughout the system and adversely affecting its safety. Therefore, through the safety monitoring technique, the DAL level assigned to a software component will be derived from the severity of the consequence of the loss of the system function associated to that component. From a "schedulability" point of view, monitors are commonly used in safety-critical operating systems for the monitoring of the time budgets assigned to each application (or task). In case an application exceeds its time budget, an event is raised which is dealt with at the application level, i.e., each system may take different measures to compensate for those violations. In DAL-B systems for instance, the system could simply provide an indication for the user (a human or another system) that the integrity of the system has been compromised. This can be the case of aeronautic navigation systems, where several redundant instruments are available to aid performing the same navigation functions. This means that in case the integrity of a certain system has been compromised, it is still possible to use the readings of another instrument that performs identical functions.

DO-178C describes three important attributes that should be considered when designing the safety monitor. The first attribute is related to the monitor DAL assignment. The safety monitoring software inherits the DAL of the highest failure severity category associated with the monitored function (similarly to the IEC61508). The second attribute is aimed at ensuring that the monitors are designed and implemented in such a way that it will detect the intended faults under all necessary conditions (otherwise it cannot be trusted and thus becomes useless). In order to ensure that all fault conditions are identified, an assessment of the system faults needs to be performed to ensure that the monitor will cover all cases. The last attribute refers to the independence between the monitoring and the monitored functions. The monitor and the protective mechanisms triggered by the events generated by the monitoring function should not be affected by the same failure causing the failure condition it is supposed to monitor. For instance, a monitor that is supposed to detect non-respected timing properties of software components (e.g., due to starvation), cannot be subject to the same source of blocking as the monitored tasks. In this case, the monitor and the monitored tasks should for example be associated to different partitions.

4.2 The IEC61508

The IEC61508 (2010) is a generic safety standard widely used throughout the industry. It serves as a common base for domain specific standards such as the ISO26262 (2011) (automotive), or EN 50128 (2009) (railway). IEC61508 is composed of a series of eight volumes addressing the complete safety life-cycle activities for systems comprised of electrical/electronic/programmable electronic (E/E/PE) elements (including software) that are used to perform safety functions. This standard defines strict rules regarding the *isolation* and *independence* between safety related and non-safety related functions. For instance:

"Where the software is to implement both safety and non-safety functions, then all of the software shall be treated as safety-related, unless adequate design measures ensure that the failures of non-safety functions cannot adversely affect safety functions." [Sect. 7.4.2.8 of IEC61508-3].

"Where the software is to implement safety functions of different safety integrity levels, then all of the software shall be treated as belonging to the highest safety integrity level, unless adequate independence between the safety functions of the different safety integrity levels can be shown in the design. It shall be demonstrated either (1) that independence is achieved by both in the spatial and temporal domains, or (2) that any violation of independence is controlled. The justification for independence shall be documented." [Sect. 7.4.2.9 of IEC61508-3].

Under IEC61508, several safety-related software design techniques and measures are presented as detailed below.

4.2.1 Partitioning

Partitioning is a technique that allows isolating software components from each other. This isolation is essential for critical systems as it allows the containment of faults, as well as the reduction of the software V&V effort. Typically, there are two approaches to achieve partitioning between software components. The first approach is to *physically* segregate the components by allocating unique hardware resources to each component (i.e., only one software component is executed on each hardware component composing the system). The second approach is to *virtually* separate the components by establishing partitioned hardware provisions that allow multiple software components to run on the same hardware platform.

Annex F of IEC61508-3 provides further recommendations on techniques for achieving non-interference between software elements on a single computer. In this context, the term "independence of execution" is used, meaning that applications should not interfere with each other's behaviour . This independence *shall* be achieved and demonstrated in both *spatial* and *temporal* domains. Spatial isolation means that one application shall not change data used by another application . Note that spatial isolation is even more important considering the fact that the highest severity software failure modes are typically associated to data corruption (e.g., due to buffer overflows or memory violation). Temporal isolation on the other hand shall ensure that one application will not cause malfunction of another application by consuming too high processor execution time or by blocking a shared resource used by other applications, thus affecting its timing properties. In order to demonstrate the independence of execution, an analysis of the proposed design is performed to determine the causes of execution interference in both spatial and temporal domain through the application of the methodologies described in Sect. 2.

The standard explicitly recommends the following techniques for achieving and demonstrating spatial independence (Sect. F.4 of IEC61508-3):

1. hardware memory protection;
2. virtual memory space;
3. rigorous design, source code and possibly object code analysis; and
4. software protection of higher integrity applications.

Ideally, data should not be passed between applications of different criticalities. However, in practice, especially in MCS, there may be a need to exchange data between applications of different criticalities. Considering this, the system should ensure that higher SIL applications are able to verify the integrity of any data received from lower SIL applications. This can be achieved, for instance, through the use of unidirectional interfaces such as messages or pipes, rather than through shared memory.

With respect to temporal independence, the following techniques (intrinsically related to the choice of scheduling policy) are mentioned by the standard (Sect. F.5 of IEC61508-3):

1. Deterministic scheduling methods such as cyclic scheduling and time triggered architectures;
2. Strict priority based scheduling by real-time executive (with mechanism to avoid priority inversion);
3. Time fences that terminate the execution of an application in case it exceeds its time budget;
4. Time slicing, which ensures that no process can be starved of CPU time.

However, the resource sharing protocol is also important when sharing resources between applications, because the design shall ensure that the applications will not malfunction due to a locked resource. Therefore, it is essential that the time required to access a shared resource is taken into consideration when performing the timing analysis of the system.

Note that the software functions used to provide spatial and/or temporal independence (e.g operating system, real-time executive) shall be allocated the *highest criticality* of the applications running on top of them (Sect. F.6 of IEC61508-3), since such software represents a potential common cause of failure of the independent elements.

4.2.2 Diverse monitor

The diverse monitor (Sect. C.3.4 of IEC61508-7) is an architectural design technique that allows the protection against faults in software, preventing the system from entering an unsafe state. It is an external monitor, running in an independent hardware, which continuously monitors the main application. In the occurrence of a fault, the monitor will trigger an event (e.g., fire an alarm) so that a corrective measure can be activated, e.g., through a restart of the monitored application or through a human operator action. Typically, the utilization of a monitor allows to reduce the criticality of the monitored application. Indeed, following the FMEA analysis of the main application (see Sect. 2), the monitor would appear as a compensating provision that would prevent the failure from propagating throughout the system, thus reducing the criticality of the monitored application. However, in this case it can be considered that the monitor "inherits" the criticality of the monitored application, because if the monitor fails, there is typically no compensating provision to compensate for that failure. Therefore, the monitor is assigned a criticality derived from the highest severity failure modes of the monitored applications. In short, if the monitor can be certified at the highest criticality level, then the criticality level of the monitored component can be reduced under the condition that an effective corrective measure is available.

4.2.3 Dynamic reconfiguration

Another architectural design technique is the dynamic reconfiguration of the system (Sect. C.3.10 of IEC61508-7), whose objective is to maintain the system functions operational despite an internal fault. This concept is more commonly applied to the recovery from hardware faults, but it can also be applied to software, if the logical architecture of the system can be mapped onto a subset of the available resources, e.g., through "run-time redundancy" to allow a software re-try or through redundant data, which can reduce the severity of the consequence of an isolated failure.

4.2.4 Graceful degradation

Graceful degradation is a technique aimed at maintaining the more important system functions available, despite failures, by dropping the less important system functions. According to the IEC61508-7, Sect. C.3.8:

"This technique gives priorities to the various system functions to be carried out by the system. The design ensures that if there is insufficient resources to carry out all the system functions, the higher priority functions are carried out in preference to the lower ones. For example, error and event logging functions may be lower priority than system control functions, in which case system control would continue if the hardware associated with error logging were to fail. Further, should the system control hardware fail, but not the error logging hardware, then the error logging hardware would take over the control function.This is predominantly applied to hardware but is applicable to the total system including software. It must be taken into account from the topmost design phase."

As it will be further discussed in Sect. 6, most of the academic works on mixed-criticality scheduling claim to implement a graceful degradation strategy. To help understand the discussion that we will have in Sect. 6 about that claim, note already a few details of the quoted example:

1. The illustrative example involves a *high priority* function and a *low priority* function (i.e., it does not refer to criticality but priority).
2. The high priority task continues to run if the low priority task fails (i.e., a failure of a low priority task does not impact on the execution of a high priority task).
3. The hardware dedicated to the low priority function is used to execute the high priority task if the hardware of the high priority task comes to fail, thus stopping the execution of the low priority task.
4. The example assumes a hardware failure, which leaves the system with not enough hardware resource to serve all the software functions.

Note also that this is the only example given in the IEC61508 standard to describe the graceful degradation concept.

4.2.5 Performance modelling

Performance modelling (Sect. C.5.20 of IEC61508-7) ensures that the system operational capacity is sufficient to meet the specified throughput and response time requirements, considering any constraint on the use of system resources. The system processes and their interactions are modelled, including their demanded resources (e.g. CPU time) under average and worst-case conditions. Performance properties such as worst-case throughput and response times of the individual system functions are then calculated. To avoid the risk of resource starvation, the systems are often designed to use only some fraction of the total available resources (e.g. 50%, as explained in Sect. C.5.20 of IEC61508-7).

4.2.6 Response timing and memory constraints

It consists in determining the temporal and memory demands under average and worst-case conditions to ensure that the system requirements will be met (Sect. C.5.22 of IEC61508-7). One of the methods to obtain these estimates is through prototyping and benchmarking of time critical systems. In terms of schedulability analysis, this is the usual analysis that needs to be performed on MCS to ensure that all safety-critical system functions will successfully meet their deadlines under the given system constraints.

4.3 The ISO26262

ISO26262 is an adaptation of IEC61508 addressing the specific needs of the automotive sector. Therefore, everything discussed in the two previous subsections is also applicable to this standard. For instance, software partitioning aspects are addressed in Sect. 7.4.11 and in Annex D of ISO26262-6. Mechanisms for error detection at the software architectural level (including monitoring techniques) are listed in Table 4 of ISO26262-6. Several techniques for temporal and logical program sequence monitoring at the hardware level are also presented in Table D.10 of ISO26262-5.

From a shared resource viewpoint, if software partitioning techniques are to be applied, the resources shared between the partitions must be used in such a way that the software components running on the different partitions do not interfere with each other.

At the software architectural design level, the standard establishes that an upper estimation of required resources for the embedded software shall be made, which includes the execution time, the storage space (e.g. RAM for stacks and heaps) and the communication resources.

Annex D of ISO26262-6 (ISO26262 2011) also provides some common examples of timing and execution faults that can cause interference between software elements of different partitions and must therefore be assessed before certifying the system: blocking of execution, deadlocks, livelocks, incorrect allocation of execution time and/or incorrect synchronization between software elements. To prevent or mitigate these faults, some mechanisms are also referred such as: cyclic execution scheduling; fixed priority based scheduling; time triggered scheduling; monitoring of processor execution time; program sequence monitoring, and arrival rate monitoring. These important aspects must be considered when designing a real-time scheduling algorithm and/or a resource sharing protocol for MCS.

4.4 Final remarks

In the previous subsections, we have presented a summary of several requirements from industrial standards that must be considered in the design of MCS. Those can be transversally applied to several domains of application (e.g. aerospace, automotive, railway). Although the presented safety-related industrial standards do not explicitly specify requirements for MCS, they do specify stringent requirements that must be met to ensure the *safety* of the system, especially in terms of isolation and independence between applications running on the same platform. Notwithstanding, the irreversible and inevitable appearance of multi-core hardware platforms in the industry introduces several additional challenges in terms of scheduling and resources sharing, which makes the isolation and independence of the mixed-criticality applications even more complex (and even more imperative). The requirements presented are clear in what concerns the isolation and independence of applications, even when they share common resources. Therefore, *when designing a scheduling algorithm and/or resource sharing protocol that is intended to be compliant with such standards, it is necessary to provide evidences that the isolation between components is sufficient to avoid failure propagation between them.* To address these challenges, several techniques have also been described that can be applied to the design of such systems, which can support the generation of the evidences required by the certification authorities.

5 Industry solutions for mixed-criticality systems

After presenting in the previous section the summary of relevant requirements from safety-related standards for the design of MCS, we present two examples of industry solutions (ARINC-653 and AUTOSAR) that meet the isolation and independence requirements established by the previously presented industrial standards. As explained by Burns and Davis (2013), most avionics and automotive complex embedded systems are evolving to mixed-criticality systems, mainly due to increasingly demanding non-functional requirements related to cost, space, weight, heat dissipation and power consumption. ARINC-653 and AUTOSAR provide the basic platform for supporting MCS in avionics and automotive domains, respectively. However, the implementation of MCS solutions based on ARINC-653 and AUTOSAR, i.e., solutions that can satisfy the stringent partitioning requirements, but still allowing for efficient sharing of resources among the partitions is still a problem not fully solved by the industry by the time this paper is being written. As explained by Burns and Davis (2013), problems such as modelling and verification, and system problems related to the development of necessary hardware and software run-time controls are still currently being addressed. Therefore, with the goal of contributing to the research aimed at the resolution of those issues, we present next the key aspects of ARINC-653 and AUTOSAR that are relevant for the development of MCS.

5.1 ARINC-653

ARINC-653 specifies the baseline operating environment for application software running on an Integrated Modular Avionics (IMA) (Watkins and Walter 2007; Diniz and Rufino 2005) platform or in traditional federated architectures developed according to ARINC 700 avionics standards (ARINC 2015).

The purpose of an IMA system is to support the execution of one or more avionics applications independently. Each application may have completely different requirements and thus be associated different DALs (i.e., criticalities). This separation is achieved through the partitioning technique, which provides the functional separation of the applications (mainly to inhibit failure propagation), as well as the facilitation of the V&V activities. A partition is basically an environment running a program, comprising its own data, context, configuration attributes, etc.

The primary objective of ARINC-653 is to define a general-purpose interface between the avionics application software and the operating system (OS) running on a avionics computer. This interface is known as the APEX (APlication/EXecutive). The APEX defines the interfaces that allow the applications software to control the OS scheduling, communication and status information functions. The key objectives of the APEX interface are portability, reusability, modularity and integration of software of multiple criticalities, i.e. it supports the co-location of software applications of different levels of criticality. However, it is important to note that the APEX interface merely defines the services that the OS needs to provide. It is up to the OS provider to implement those services.

Under ARINC-653, it is required that the hardware offers the following functionalities to the OS: ability to restrict memory spaces, processing time and access to I/O for each individual partition.

Although industrial solutions such as ARINC-653 exist, most of them were initially intended for single-core platforms and must now be extended to multi-core.

The following subsections briefly describe some key functionalities specified by ARINC-653 that are pertinent to this work.

5.1.1 Partition management

The partitioning concept is central to the ARINC-653 philosophy, whereby the programs resident on the partitions are partitioned with respect to space (memory partitioning) and time (temporal partitioning). The partitioned system has to be robust enough to support applications of different criticality levels to execute in the same core platform, without affecting each other, both spatially and temporally.

The scheduling of partitions shall be strictly deterministic over time. Based on the configuration of the partitions, time windows are assigned to each partition which are then activated on a time-based schedule. The schedule is fixed for a particular configuration. Partitions are scheduled on a fixed, cyclic basis. This provides a deterministic scheduling methodology whereby partitions are assigned a fixed amount of CPU time. This ensures that each partition will have uninterrupted access to common resources during its assigned period. The scheduling configuration is done by the system integrator only and thus cannot be modified during operation. The memory areas allocated to each partition are predefined. Access outside each partition's assigned memory area is forbidden.

5.1.2 Process management

In ARINC-653, the term "process" is used to designate each "task" comprised within a partition. Still, the term task is more commonly adopted within the real-time community and it is the term chosen to be used in this paper.

Considering the IMA concept, within each partition, the scheduling model defines the tasks as the scheduling units. One or more tasks can operate concurrently to provide the aeronautic functions required by the partition. A fixed priority is assigned to each task. A task can be preempted at any time by another with higher priority. The tasks may be periodic or sporadic and certain scheduling capabilities are required by the OS to accurately control the tasks execution in order to meet the application timing requirements. These tasks interact with the OS through the APEX interfaces and the OS is responsible to arbitrate the tasks access to the CPU. The occurrence of a fault may trigger an action to either initialize or terminate a task. Therefore, a method is required to safely synchronize the access to the system's mutually exclusive resources. The partition code executes in *user mode* only, i.e. no privileged instructions are allowed. It is also important to highlight that the above concepts are applicable to the tasks inside a partition, i.e., tasks that have the same criticality level. Hence, the decision to terminate a task in case of misbehaviour is not linked with its criticality, but rather with the application requirements.

5.1.3 Memory allocation

The partitions and their associated memory spaces are defined during system configuration, i.e. offline. The APEX interface does not offer memory allocation services. The exchange of information between tasks pertaining to different partitions can be done through buffers, which are also declared at the system configuration.

5.1.4 Intrapartition communication

Intrapartition communication is performed by means of buffers, blackboards, semaphores, and events. Buffers and blackboards are used for general inter-process communication, whereas semaphores and events are used for inter-process synchronization. Buffers and blackboards allow processes to communicate by exchanging messages that are directed to the respective buffers and blackboards associated with the destination processes. The memory space required to manage buffers and blackboards and to store messages is allocated for the partitions memory and is configured at design time. Buffers store multiple messages in message queues, either in FIFO or priority order. Blackboards on the other hand do not allow message queueing. Messages written to a blackboard remain there until they are either cleared or overwritten by a new instance of the message.

5.1.5 Intepartition communication

Interpartition communication is conducted via the exchange of *messages* through a *channel*. A message is defined as a continuous block of data of finite length. The channels define a logical link between a source partition and one or more destination partitions. The partitions can access the channels via defined access points designated as ports. The ports provide the required resources that allow a partition to send and receive messages over a specific channel. It is important to note that the destination of a message is always a partition, and not a process within a partition. As for intrapartition communications, interpartition communication is also performed with queues and blackboards.

5.1.6 Time management

The OS provides time slicing for partition scheduling, deadline, periodicity, and delays for task scheduling. Timeouts for intrapartition and interpartition communication are also provided in order to manage time. Time is unique and independent of partition execution. A time capacity is associated with each task that ensures that its processing requirements are met. When a task starts, its deadline is set to the value of the current time plus an execution time capacity.

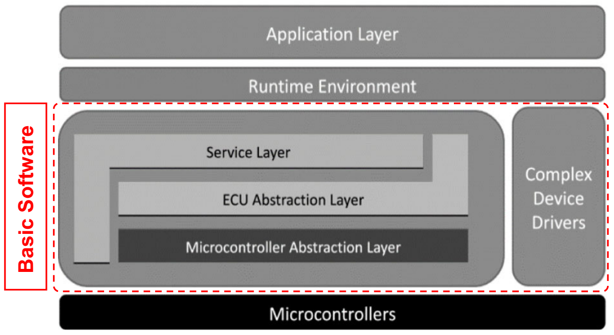


Fig. 2 AUTOSAR architectural overview (AUTOSAR 2011)

5.2 AUTOSAR

AUTOSAR appears as a response of the automotive industry OEMs (Original Equipment Manufacturers) to the increase of complexity on vehicle applications. It is a standardized and open software architecture that can be used for the diverse in-vehicle systems without compromising the quality and with cost-efficiency. The software that implements the automotive functionality is mainly encapsulated in software components. The standardized functional interfaces (with different layers) of those components are a central element to support scalability and transferability of automotive functions across electronic control units of different vehicle platforms. AUTOSAR achieves the standardization of functional interfaces across manufacturers and suppliers and also the standardization of interfaces between different software layers. Hardware abstraction layers are present upon these, besides the software specific ones.

The standard scope includes all vehicle domains, and serves as a platform upon which vehicle applications can be implemented. This process aims at minimizing the barriers between the functional domain and the mapping of functions and functional networks, almost independently of the associated hardware. The illustration presented in Fig. 2 describes an AUTOSAR architectural overview.

By looking at Fig. 2 bottom-up, the following layers are identified:

– Basic Software (BSW): The standardized software layer that provides the services to the AUTOSAR application software components (functional part), including the communication with the hardware. The functional part needs the basic software layer to run, but the BSW layer does not perform any functional job itself. The BSW is composed of the following sub-layers: Complex Device Drivers (CDD), hardware abstraction (ECU[4] and Microcontroller) and AUTOSAR Services. The CDD is a loosely coupled container, where specific software implementations can be placed. The main goal of the CDD is to provide support for complex sensors and actuators that have special functional and timing requirements. It might also be used to encapsulate legacy functionality of a non-AUTOSAR system (AUTOSAR 2015). The Microcontroller abstraction layer (MCAL) provides a standard inter-

4 ECU: Electronic Control Unit

face to the components of the BSW, which avoids direct access to microcontroller registers from higher-level software (AUTOSAR 2011). The ECU abstraction provides the software interface to the specific ECU electrical interfaces, thus removing all the higher-level software dependencies from the underlying hardware. The Services sub-layer is split into three main services: general services (e.g. diagnostic protocols and memory management), communications (e.g. CAN), operating system services (e.g. priority-based scheduling).

– Runtime Environment (RTE): RTE implements the communication functions for inter- and intra-ECU information exchange. It provides communication abstraction to the software components attached to it at the application layer, which is independent from whether inter-ECU communication channels (e.g. CAN) are used or communications are intra-ECU. RTEs are ECU specific and need to be tailored according to the communication requirements of the software components running on top of them. Note that in the AUTOSAR concept, the RTE is the concretization of the *Virtual Functional Bus (VFB)*. The VFB is an abstraction layer that allows the communication between different AUTOSAR Software Components and its environment (e.g. hardware driver, OS, services, etc.) to be specified independently from any underlying hardware (e.g. a communication system). The advantage of this approach is that it enables the virtual integration of different automotive software components very early in the design process.
– Software Component (SWC): the application layer is composed of interconnected "AUTOSAR Software Components". The AUTOSAR SWCs are atomic pieces of software that implement part of an application, are independent of the infrastructure, and can be mapped on an ECU. Each Software Component within an ECU encapsulates a distinct part of the overall application functionality. SWCs expose well-defined interfaces, described and standardized within AUTOSAR.

The AUTOSAR OS standard specifies a fixed priority preemptive RTOS with support for multi-core. Some aspects of the AUTOSAR OS that are relevant to the support of mixed-criticality are discussed in the following subsections.

5.2.1 Time partitioning

AUTOSAR requirement SRS_Os_11008 (AUTOSAR 2013) specifies that the OS shall not allow a timing fault in any OS-Application to propagate to other applications running on the same processor. A timing fault of a task or interrupt service routing (ISR) occurs when: (i) the specified execution time is violated or (ii) the specified arrival rate is violated. To ensure that a task or ISR meets its deadline in a fixed priority preemptive operating system like AUTOSAR OS, the following protections are necessary:

– Prevention of timing errors from the execution time of Task/ISRs in the system by using *execution time protection* to guarantee a statically configured upper bound, called the Execution Budget;
– Prevention of timing errors from the blocking time that tasks/ISRs suffers from lower priority tasks/ISRs locking shared resources or disabling interrupts by using

locking time protection to guarantee a statically configured upper bound, called the Lock Budget;
- Prevention of timing errors from the inter-arrival rate of task/ISRs in the system by using inter-arrival time protection to guarantee a statically configured lower bound, called the Time Frame;

5.2.2 Space partitioning

AUTOSAR requirement SRS_Os_11005 (AUTOSAR 2013) specifies that the OS shall provide the ability of partitioning OS-Applications with respect to memory and prevent an OS-Application from modifying the memory of other OS-Applications. However, memory protection under AUTOSAR is only possible on processors that provide hardware support for memory protection. The memory protection scheme is based on the (data, code and stack) sections of the executable program, as described next:

- Stack protection: an OS-Application comprises a number of tasks and ISRs. Memory protection for the stacks of tasks and ISRs is important because it:
  - provides a more immediate detection of stack overflow and underflow for the task or ISR than can be achieved with stack monitoring;
  - provides protection between constituent parts of an OS-Application, for example to satisfy some safety constraints;
- Data protection: OS-Applications can have private data sections and tasks/ISRs can have private data sections. OS-Application's private data sections are shared by all tasks/ISRs belonging to that OS-Application;
- Code protection: code sections are either private to an OS-Application or can be shared between all OS-Applications (to use shared libraries). In the case where code protection is not used, executing incorrect code will eventually result in a memory, timing, or service violation.

5.2.3 Communications between applications

Since AUTOSAR requires that OS-Applications be protected against each other, a mechanism needs to be provided to transport data between those applications. AUTOSAR offers a communication mechanism to transfer data between OS-Applications: the Inter OS-Application Communicator (IOC).

The IOC is part of the operating system and is responsible for the communication between OS-Applications and in particular for the communication crossing core or memory protection boundaries. The IOC is a third type of communication, in addition to: intra OS-Application communication (always handled within the RTE) and inter ECU communication [already available via well defined interfaces to the communication stack (COM)]. The IOC offers communication of data to another core or between memory protected partitions with guarantee of data consistency. The IOC provides communication buffers, queues, and protected access functions/macros to these buffers that can be used from any pre-configured partitions concurrently.

5.3 Final remarks

After having analysed the key aspects of the industry solutions for implementing mixed-criticality systems, there are some important aspects that need to be highlighted. Although the ARINC-653 and AUTOSAR approaches have some commonalities, these standards propose very different techniques to solve the issues of independence and isolation between applications. In ARINC-653 this goal is achieved through hierarchical scheduling, where partitions are scheduled using a table driven approach, i.e., no changes can be made to the schedule during run-time, hence resulting in a system with reduced flexibility. This scheduling policy renders the isolation extremely robust against any timing fault. However, it also involves a complex configuration phase where partitions must be dimensioned and the partition schedule compiled. System integrators must consider the timing constraints of each independent task along with the interactions between all the different partitions, I/O devices and other shared resources. AUTOSAR on the other hand proposes a more flexible approach, which does not specify a specific scheduling policy, and is thus more open (or less conservative); note, however, that AUTOSAR still specifies timing protection requirements that must be provided to prevent a timing fault in any OS-Application to propagate to a different application resident in the same processor. ARINC-653 and AUTOSAR have similar requirements with respect to spatial partitioning and in both cases, the communication between partitions/applications must be made through a protective layer, i.e., a mechanism to transport data between applications. ARINC-653 uses messages and channels, and AUTOSAR uses the IOC.

6 The theoretical MC model and its certification concerns

In the previous sections we have presented the main design principles and requirements that drive the development of industrial MCS. We now move the focus of the paper to the academic work. We will discuss one of the most prominent model adopted in the state-of-the-art and some common misconceptions.

6.1 The state-of-the-art in academy

These last years, the real-time research community has been extremely active in the domain of MCS. Almost 200 papers treating of the scheduling of MCS have been referenced in Burns and Davis (2013), and tens of related papers are still published every year. It would therefore be unrealistic to review and analyse here the whole state-of-the-art on real-time scheduling of MCS. Instead, this section evaluates the key concepts and approaches commonly encountered in real-time scheduling models and algorithms against the recommendations and requirements found in the safety-related industrial standards that were presented in the previous sections.

Most of the works about MCS published by the real-time scheduling research community are based on a model proposed by Vestal (2007). Therefore, we will mainly focus on that specific model and its related work. Yet, the discussion proposed in the rest of this chapter is quite generic and most of the remarks made are thus valid for other academic MC models introduced during the last decade. The Vestal model assumes that the system has several modes of execution, say modes $1, 2, \ldots, L$. The application system is a set of real-time tasks, where each task τ_i is characterized by a period and a deadline (as in the usual real-time task model), an assurance level ℓ_i and a set of worst-case execution time (WCET) estimates $\{C_{i,1}, C_{i,2}, \ldots, C_{i,\ell_i}\}$, under the assumption that $C_{i,1} \leq C_{i,2} \leq \cdots \leq C_{i,\ell_i}$. The different WCET estimates are meant to model estimations of the WCET at different assurance levels. The worst time observed during tests of normal operational scenarios might be used as $C_{i,1}$ whereas at each higher assurance level the subsequent estimates $C_{i,2}, \ldots, C_{i,\ell_i}$ are assumed to be obtained by more conservative WCET analysis techniques or by considering safety margins imposed by certification authorities. Vestal initially studied the schedulability of the system under such assumptions and proposed a task priority assignment algorithm optimising the overall schedulability. In later works, solutions started to be proposed in order to improve the schedulability of higher criticality tasks over the lower criticality ones. Those solutions usually rely on variations of the following scheme (Baruah et al. 2011). The system starts its execution in mode 1 and all the tasks are scheduled to execute on the core[s]. Then at runtime, if the system is running in mode k then each time the execution budget $C_{i,k}$ of a task τ_i is overshot, the system switches to mode $k+1$. It results from this transition from mode k to mode $k+1$ that all the tasks of criticality not greater than k (i.e., $\ell_i \geq k$) are suspended. Mechanisms have also been proposed to eventually re-activate the dropped tasks at some later points in time (Santy et al. 2013).

It must be noted that one of the derivatives/simplifications of this model is the Vestal's model with only two modes, usually referred to as LO and HI modes (which stand for Low- and High-criticality modes). Multiple variations of that scheduling scheme exist (please refer to Burns and Davis (2013) for a comprehensive survey); some for single-core, others for multi-core architectures. In the case of multi-core, both global and partitioned scheduling techniques have been studied. Solutions for fixed priority scheduling, earliest deadline first, and time-triggered scheduling have been proposed. Some works also propose to change the priorities or the periods of the tasks during a mode change rather than simply stopping the less critical ones. Note that we use Vestal's model to illustrate this section as most (not all) theoretical works in the state-of-the-art are based on similar assumptions. However, as mentioned in Burns and Davis (2013), more practical solutions such as the EMC2 framework (Chisholm et al. 2016; Kim et al. 2016; Chisholm et al. 2015) are also being developed in the context of academic research. Even if those works are not directly mentioned here, we believe that some of the concepts discussed in this section may be useful to back up choices made in those more applied researches.

6.2 The mismatch in the notion of criticality

There is a clear mismatch of interpretation of the concept of "system criticality" between the industrial standards and the academic papers based on Vestal's model (Vestal 2007), which use the terminology "system criticality" to refer to modes of execution of software tasks (e.g., high or low criticality). That is, switching from a mode k to a mode $k+1$ is usually referred to as an "increase of the system criticality level". Although this concept is not fundamentally wrong, it creates confusion in the context of industrial MCS, where the term "system criticality" is used to refer to the level of assurance (DAL or SIL or ASIL) applied in the development of a software application that implements critical system functions, i.e., safety functions.

Therefore, there exist a misalignment in the interpretation of the notion of critically by the MCS scientific literature and the safety-related industrial standards. We believe that over the years this discrepancy has generated some sort of confusion, which caused the academic work not to be fully compliant with the concept of criticality defined in the standards. In order to address this issue, we warmly invite the reader to consult the part 1 of the ISO26262 that clearly defines fundamental concepts that pertain to safety-critical systems, such as "safety functions", "safety-related systems", and "safety integrity level".

6.3 Software task assurance level and the notion of importance

As a second point, in the standards the word "function" is used at the system level, in reference to a system function, or in other words, an action that the system must be able to perform (accelerate, break, etc.). A "function" as defined in the standard may thus involve the whole chain of software and hardware components that play a role in the execution of that action. It may include sensors, processing elements, and actuators for instance, according to IEC61508. This implies the assignment of a SIL to the whole functionality and not only the individual software functions that are part of it. This is explicitly written in the following note associated to item 3.5.10 of IEC61508-4 (IEC61508 2010):

"SIL characterises the overall safety function, but not any of the distinct subsystems or elements that support that safety function. In common with any element, software therefore has no SIL in its own right. However, it is convenient to talk about "SIL N software" meaning "software in which confidence is justified (expressed on a scale of 1 to 4) that the (software) element safety function will not fail due to relevant systematic failure mechanisms when the (software) element is applied in accordance with the instructions specified in the compliant item safety manual for the element".

The concept of SIL is not trivial and its definition is commonly misunderstood. It is common to think that a real-time task of higher SIL is "more important" than a task of lower SIL (we further discuss that point in the next section). In actual systems these tasks are part of one or several system functions and those functions are the entities which are assigned a SIL. Therefore, a real-time task must be implemented in accordance with the development rules defined for the SIL of the functionality to which it belongs. If the task belongs to more than one functionality then it must naturally be implemented in accordance with the development rules defined for the highest SIL among the SILs of all the functionality to which it belongs. Once implemented, the SIL of the functionality to which the software function belongs will also impact the way the function will be deployed on the hardware architecture and it will define or restrict its interaction with other functions.

For example, the implementation of a system function *A* that has to be conform with SIL 4 is more costly than the development of the same system function in accordance to SIL 3. But by no means this implies that a system function *A* implemented at SIL 4 is more important than any other system function *B* implemented in accordance to SIL 3. Irrespective of their SIL, *all these functions are safety-critical* and may cause severe damage in case of failure. Therefore, a software function that is part of a high-SIL functionality cannot be considered as "more important" than a software function that is part of a functionality of lower SIL. Moreover, the notion of "importance" is not explicitly defined in the safety-related standards, thus it can be considered as subjective and/or ambiguous. Only an application designer is qualified to define which functions are important and which ones are not, depending on the mode of operation of the system. Therefore, a unique criteria for defining "importance" is very difficult to establish, because it fully depends on the specific application context.

In order to further illustrate the difference between the notions of importance and criticality, and also to illustrate why the "discarding tasks model" discussed in the previous subsections is not the most appropriate degradation model for safety-critical systems, a more specific example from the automotive sector is provided.

Example 1 The example is composed of two configuration scenarios, each one of them with two fictitious automobiles: AUTO1 and AUTO2. Consider two system functions SF1 and SF2 implemented in each car, respectively. Let us also assume that in case of a serious failure of SF1 and SF2, both cars will be unusable, but in the case of AUTO1 the driver is able to control the car and park it safely, whereas in the case of AUTO2 the driver is not able to control the car, thus resulting in a serious crash. In both cases we assume that the probability of failure of the functions are the same. Therefore, by analysing the two scenarios involving cars AUTO1 and AUTO2, we can conclude that both functions are *important*, because both of them can potentially put the physical integrity of the passengers at risk (i.e. harm) in case of failure. Let us assume that a risk assessment was performed for each scenario to determine the *criticality* (or ASIL) of functions SF1 and SF2. As previously shown in Table 1, the ASIL resulting from the risk assessment is a function of three parameters: probability of exposure, the controllability, and the severity of the hazardous events with regard to the item under analysis, i.e., SF1 and SF2 in our example. Now let us assume that as a result of the risk assessment, SF1 was assigned with severity class S3,[5] probability class E4[6] and controllability class C2,[7] thus leading to the assignment of ASIL C. SF2 was also assigned with severity class S3, probability class E4, but with controllability class C3,[8] thus leading to the assignment of ASIL D. As described in this example, even though SF1 and SF2 have the same severity and probability classes, they differ in the controllability classes, i.e., when SF1 fails, the driver is normally able to control the vehicle, whereas in case SF2 fails, the driver is not able to control it. Through this example, we can highlight the following important observations:

[5] S3: life-threatening injuries (survival uncertain), fatal injuries;

[6] E4: high probability;

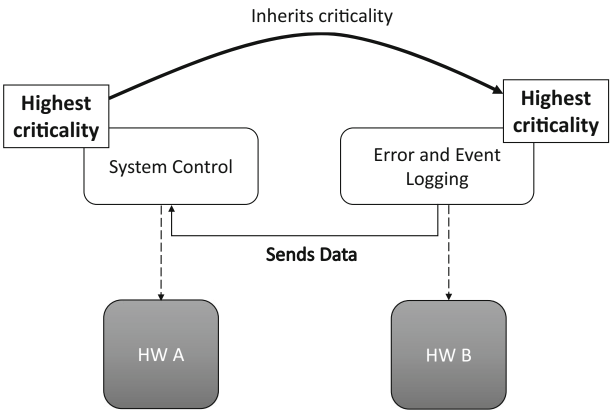
[7] C2: normally controllable;

[8] C3: difficult to control or uncontrollable.

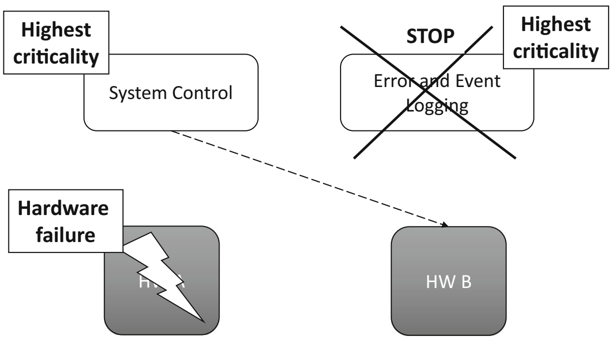
- The software tasks that implement the automotive function SF2 cannot be considered more important than the software tasks that implement the automotive function SF1, because both lead to failure conditions with the same severity class. Hence, the criticality of each task is not assigned just as a function of its importance for the system, the probability of occurrence, the severity of its failure, or the capacity of the system to recover from its loss. It is instead the result of an analysis combining all those different factors (e.g. by combining FMEA and FTA) that will define the criticality, and consequently all the requirements and rules that will have to be respected during the development process;
- Several academic works have followed the approach of associating a lower priority to lower criticality tasks, which introduces limitations to the model, because lower criticality tasks also implement functions whose effects can cause harm to the system users in case of failure;
- In the definition of a MCS model with certification potential for use in the industry, we have to dissociate the concepts of importance and criticality; as shown in the example, sometimes it is not always acceptable from the system perspective to stop (either permanently or temporarily) a less critical task in favour of a critical task with higher criticality, because both of them are important to ensure the safety of the system. Therefore, from a certification point of view, it would be extremely difficult to justify and demonstrate the scenarios under which it would be acceptable to stop a lower criticality task in favour of a high criticality one.

Example 2 Now let's discuss even further the notions of importance and criticality by analysing a generic industrial scenario, based on the example from IEC61508-7 provided in Sect. 4.2.4 of this paper and illustrated in Fig. 3. In that example we have two system functions: a system control function (SC1) whose software runs on its associated hardware (HW A) and an error and logging function (EL1) whose software runs on another hardware (HW B). These two functions are clearly not isolated because they exchange data between each other (Fig. 3a). Normally EL1 sends data to SC1 in order to improve the quality of control of the system. The system control function is typically assigned the highest criticality level applicable to the system, because it implements essential functions that in case of failures can lead to severe consequences. Due to the fact that EL1 is not isolated from SC1, it will inherit the criticality of SC1, i.e., the highest level. When a failure in HW A occurs (Fig. 3b), the system can be reconfigured, by stopping EL1 (non-essential function) and switching the execution of SC1 to HW B. Through this example, we can highlight the following important observations:

- Contrary to the approach adopted by the Vestal model, the example illustrates that in some scenarios it is possible sometimes to suspend a high criticality task of lower importance in favour of a task of the same criticality level with higher importance. Therefore, provided that the safety analysis (e.g. FMEA) shows that there will be no safety consequences to the system, it is possible sometimes to stop a high criticality task;
- In safety-critical systems, the decision of which task can be stopped during run-time must be taken during design time and must rely on the risk assessment



(a) Logging function: high criticality but lower importance



(b) Stopping a high criticality function (logging)

Fig. 3 Example of a generic industrial scenario

performed, which will map beforehand all the possible events that can trigger such suspension. In the example provided, the conditions to be met (e.g. hardware failure) that allow the logging function to be stopped are typically identified during design time, based on the analysis of this potential failure mode (e.g. by means of a FMEA).

To conclude, different SILs (or ASILs) do not mean different importance. Tasks of different SILs are simply subject to different development requirements but the isolation and independence between them still has to be preserved and guaranteed to ensure safety. Therefore, the conditions that can trigger the suspension of some task in favour of another must always be identified during design time and justified by a risk assessment. Note, however, that the approach adopted by the Vestal model works fine for dual criticality systems where one can have critical tasks (e.g., DAL A, B or C, or ASIL D, C or B) and non critical tasks (e.g., DAL D or E or ASIL QM) in the same system. In that case, low criticality tasks are also less important than high criticality ones. The equivalence disappears when a system is composed of tasks of several criticality levels (e.g. DAL A, B and C, or ASIL D, C and B). In that latter case, a DAL C task is not necessarily less important than a DAL B task.

6.4 Assigning different WCET estimates

Different SIL imposes different development rules, including coding rules. Although the consequences of timing violations could potentially be less severe for tasks of applications of lower SIL (yet, not always as discussed in Sect. 6.3), there is no recommendations in the safety-related standards that advocate the use of specific WCET estimation techniques. The standards simply recommends more rigorous testing methods for higher SILs to demonstrate the timing performance of the safety mechanisms at the system level (see Table 11 of ISO26262-4). In contrast, the paper from Vestal (2007) and all the academic works based on it assume that the higher the degree of assurance of a task, the more pessimistic the estimation of its WCET. Although this assumption is perfectly relevant (since a more pessimistic estimate may be understood as more reliable), it does not equate to the recommendations of the standards. More rigorous testing could also be understood as a method to achieve *less* pessimistic estimates (according to the usual trade-off between accuracy and runtime complexity of most of computation techniques). As shown in the automotive example in the previous section, it would not be an appropriate design decision to apply a greater WCET margin to the tasks that implement the automotive function SF1 (ASIL C), just because it has a lower ASIL than automotive function SF2 (ASIL D). As demonstrated in that example, both functions have the potential to cause harm to the car passengers in the occurrence of a hazardous event with the same probability level. Therefore, from the safety perspective such design option would be very difficult to justify toward the certification authority. Furthermore, one could even say that by applying a lower WCET margin to the tasks that implement the lower ASIL function as done in the Vestal model, we would be in fact increasing the probability of failure of the function of lower ASIL in relation to the higher ASIL function (through increasing the probability of overshooting its time budget), which would require a re-evaluation of the risk assessment matrix of the system.

Although this strategy for determining the WCET is valid in the conjecture of Vestal's paper, the most important aspect from the safety point of view that needs to be considered is that the accurate determination of the WCET upper-bound is a necessary but not sufficient condition to ensure the safety of the overall system (see further discussion in Sect. 8.6). In addition to that, mechanisms must be implemented to handle a task overshooting its execution budget without impacting on the system safety.

Exceeding the allocated budget is an obvious failure condition identified during the FMEA of any real-time embedded system. This failure condition can lead to a system failure that can potentially trigger a system hazard. Therefore, more important than accurate estimations of the WCET is the design of mechanisms to ensure that the system safety is not compromised in case of an excessive use of processor resources. In the next section we discuss some of the techniques proposed in the literature for handling those budget violations.

6.5 Graceful degradation

The plethora of work based on Vestal's model could be understood by its resemblance with the graceful degradation technique described in IEC61508. However, as discussed in Sect. 4.2.4 and illustrated in Example 2, this technique aims at dropping the less important system functions, and not the ones with lower criticality. Hence, graceful degradation allows the system to enter a degraded mode of operation where limited service is provided, without compromising the safety or survival of the system.

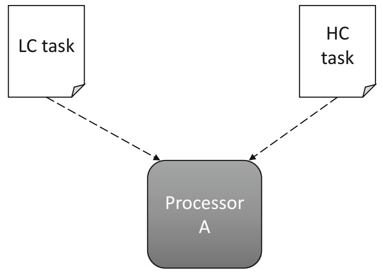
Graceful degradation is a technique that lends itself very nicely for the scheduling of MCS. However, in the context of MCS, the decision to suspend a less important task in favour of another more important one in the event of a critical fault (e.g., budget overshooting) must be supported by analyses performed during design time, including a risk assessment.

Considering the above explanation and all the concepts presented so far, it becomes clear that a safety-critical system does not "change" its criticality during operation. In practice, what can happen is that a system, during certain specific operational conditions (e.g. in case of a major failure), can enter a certain mode of operation where some or all of the non-essential system functions, i.e., those that are not required to ensure the safety and survival of the system, may be stopped, and potentially reactivated later. This principle is commonly applied in the space sector, for instance. Whenever a satellite has a major failure it may enter the so called *safe mode*, where only the essential functions (e.g platform survival functions such as the power and communication subsystems) are kept active, whilst other non-essential functions (e.g. scientific instrument data collection) are deactivated.

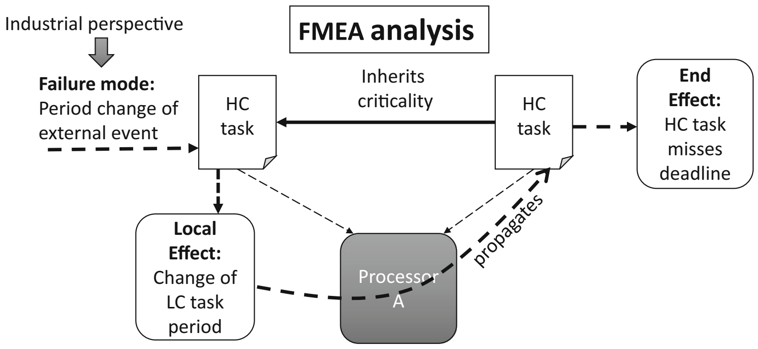
6.6 Timing isolation

Most of the theoretical works previously cited are mainly concerned about fulfilling the tasks' requirements in terms of execution time based on accurate WCET estimation techniques, as discussed in Sect. 6.4. However, if we aim towards a MCS model with certification potential, there are also other aspects that need to be considered to achieve timing isolation between applications. For instance, as stated by AUTOSAR (see Sect. 5.2), the period of activation of a task is an important parameter that needs to be taken into account. A violation of the assumptions associated to this parameter could cause the propagation of faults that could affect the timing properties of other applications residing in the same processor. This is, in fact, one of the limitations of the Vestal model, which assumes that the minimal inter-arrival times are always respected. However, this may not always be the case in industrial systems, which in many cases have to withstand very demanding operational conditions. Therefore, it becomes clear that a mechanism to encapsulate the applications is required to ensure their timing properties under all operational scenarios (i.e., nominal and non-nominal).

In addition, AUTOSAR allows tasks of different criticality to execute on the same platform, provided that a higher-criticality task gets a higher priority than a lower-criticality task, and provided that inversion effects are excluded (Ernst and Natale



(a) High and low criticality tasks scheduled on the same processor


FMEA analysis
Industrial perspective
Failure mode: Period change of external event
HC task
Inherits criticality
HC task
End Effect: HC task misses deadline
Local Effect: Change of LC task period
Processor A
propagates

(b) Example of application of an industrial FMEA

Fig. 4 Analysis of tasks isolation in the Vestal model

2016). In that case timing isolation is ensured, since a lower-priority task cannot delay a high-priority one. However, the typical scenario of most of the theoretical works previously cited that are based on the Vestal model do not enforce this restriction, thus violating the timing isolation. An additional example is provided to illustrate this scenario.

Example 3 Our example scenario is composed of a single-core processor and a scheduler running two tasks with different criticalities, one of them being the low criticality task (LC) and the other the high criticality task (HC), as shown in Fig. 4a. By adopting a typical safety-analysis industrial approach to identify and evaluate systematic failures, we perform a simple Failure Modes and Effects Analysis (FMEA) on this system by injecting a potential fault in the LC task, according to the process described in Sect. 2, and shown in Fig. 4b. It is important to highlight that when applying this process we try to brake some fundamental assumptions about the operational scenario. The failure mode we analyse in this example is a sudden change in the period of the external event that activates the LC task. This event is supposed to be periodic or sporadic, but for some reason the reaction time becomes uncontrolled and the event is activated more often than it is supposed to. As a consequence, the immediate local effect is an unexpected change in the period of the LC task. Due to the fact that the two tasks run on the same core and are scheduled by the same scheduler that does not implement any type of time partitioning, the change of period of the LC task may alter the response time of the HC task (if the LC task has a higher priority than the HC task, which is permitted in the model). In the FMEA analysis, the failure modes are evaluated based on a worst case scenario and in this case, the HC task would miss its deadline, thus leading to a failure of the highest severity applicable to the system under analysis. Since isolation between LC and HC cannot be ensured, the LC task would inherit the criticality of the HC task and both of them would need to be developed to the highest level of assurance of the system, missing the initial objective of having a mixed-criticality system.

As explained in Example 3, the most commonly adopted MCS theoretical model raises concerns in terms of tasks isolation. Therefore, in the safety-critical domain, it will be extremely difficult to prove to the certification authorities that the timing properties of higher criticality tasks will hold even under the most unfavourable scenarios. Note that if the Vestal model did impose the restriction of assigning higher priorities to higher-criticality tasks, then there would not be any justification for dropping lower criticality tasks, since they could not delay the higher-criticality ones anyway.

On the other hand, this model can potentially be very useful for the development of systems that are not safety-critical (e.g. some types of mission critical or non-critical systems—see Sect. 2.3) and that don't have so strict requirements in terms of timing isolation as required by the safety-related standards. In a non safety-related system, the idea of suspending less important tasks in favour of more important ones is perfectly acceptable, since in a worst case, only the quality of the service provided to the end users will experience some level of degradation. Hence, a possible application of that model would be to ensure the quality of service (QoS) of a non-safety-critical system, where it would be perfectly acceptable to suspend less important tasks of the system in favour of more important ones.

In conclusion, several other parameters than the WCET need to be considered when designing MCS with certification potential. The theoretical MCS model commonly adopted has several limitations in terms of timing isolation of tasks developed with different levels of assurance. An industrial MCS should be capable of mitigating or isolating any timing fault. However, the model can still be useful for the development of non safety-related systems, such as the ones that need to ensure different levels of QoS of tasks with different priorities (e.g., mission-critical or non-critical-systems).

7 Desirable requirements of a MCS model

In the previous sections we have reviewed the MCS concept and requirements in light of the safety-related industrial standards; although those standards do not always clearly and explicitly specify the requirements for mixed-criticality systems, we have researched and analysed the requirements and techniques specified by those standards that are applicable to the development of MCS; we have also discussed several misconceptions that currently exist in the real-time system scheduling models and algorithms. Considering all that previous discussion, and aiming to fill this gap between the MCS concept and the safety-related standards, we have compiled a brief set of essential requirements that define the key desirable functions and characteristics towards a MCS model with enhanced certification potential.

7.1 Time partitioning

TP.01—time budget

Real-time tasks shall shall not be allowed to exceed their reserved time budget (CPU time).
Note: In a MCS it is possible to have either "critical" or "non-critical" real-time tasks. The term "critical" is used here to denote the tasks of the system that have safety implications, as determined by the project's risk assessment process and criteria. Nevertheless, the requirement is still applicable to both categories. Note also that no specific technique to ensure temporal independence of the tasks (e.g. scheduling policy) is purposely mentioned to keep the requirement as generic as possible.

TP.02—inter-arrival rate

Real-time tasks shall respect their inter-arrival rates.
Note: In order to ensure that the tasks can meet their deadlines, the operating system must control at runtime the tasks inter-arrival rates, time budget (see TP.01) and blocking times (see SR.01 and SR.02).

TP.03—protection against timing failure propagation

In case a real-time task malfunctions (i.e. does not fulfils requirements TP.01 and TP.02), either by consuming its reserved time budget without meeting its deadline, or by suffering from over-activation, it shall not affect the timing properties of other tasks running in other partitions.
Note: The detection of such failures shall be performed by the safety monitoring function described in Sect. 7.6.

TP.05—time partitioning software development assurance level

The software that implements the time partitioning function shall be developed with the highest level of assurance required by the system, i.e. it shall be assigned the highest applicable criticality level.
Requirement Rationale: It is foreseen that this software function will normally be implemented by the scheduler.

7.2 Space partitioning

SP.01—logical isolation

All system tasks shall respect their own assigned memory space and shall not be allowed to modify data in the memory space assigned to other tasks.
Requirement Rationale: Isolation is required to protect at least the tasks data, code and stack.

SP.02—hardware enforced protection

In order to ensure the isolation of the multiple individual partitions, the underlying hardware shall be able to: restrict memory spaces, processing time, and access to input/output (I/O).

SP.03—memory space definition

The configuration of the memory partitions and their respective sizes shall be defined at design time.
Requirement Rationale: For safety reasons the configuration of the memory partitions shall not be modified at runtime and are hence defined offline.

7.3 Handling of shared resources

SR.01—intra-partition shared resources

A resource sharing protocol shall ensure that the time required to access resources that are shared within a partition is deterministic and predictable.
Requirement Rationale: An upper bound for the blocking time when accessing shared resources within a partition shall be determined and taken into account during the schedulability analysis of the tasks within a partition.

SR.02—inter-partition shared resources

A resource sharing protocol shall ensure that the time required to access resources that are shared between partitions is deterministic and predictable.
Requirement Rationale: An upper bound for the blocking time when accessing inter-partition shared resources shall be determined and taken into account during the schedulability analysis of the time partitions.

SR.03—budget expiration before resource access

A task whose budget expires while waiting for a shared resource shall not be allowed to request access again until its budget is replenished.
Requirement Rationale: Otherwise the other waiting tasks would suffer an unacceptable blocking time.

7.4 Management of modes of operation

MO.01—change of modes of operation

The system shall be allowed to change between different modes of operation without compromising the safety of the system.
Requirement Rationale: Different modes of operation are normally required by mixed-criticality systems in order for the system to able to react to external events (e.g. sensor reading) and internal events (e.g. hardware failure) in accordance with the project requirements. However, the safety properties of the system must be ensured at all

times (including the timing properties of the tasks), by ensuring that the essential critical system functions remain operational under all circumstances.

7.5 Graceful degradation

GD.01—safe graceful degradation

The system shall be allowed to change to degraded modes of operation by suspending only the non-essential tasks required for safe operation.
Requirement Rationale: The operation in degraded mode may be required by the system in case a critical fault occurs. However, only tasks that don't have safety implications (less important) are allowed to be suspended.

GD.02—task suspension

The definition of which tasks may be allowed to be suspended during runtime shall be defined during design time.
Requirement Rationale: In the event of a critical fault that triggers a transition to a degraded mode of operation, the selection of which tasks will be suspended shall have been pre-defined during design time.

7.6 Safety monitoring

SM.01—fault analysis

A fault analysis shall be performed to ensure that all potential failure conditions that are required to be monitored are identified.

SM.02—monitoring function assurance level

The monitoring function (hardware and software) shall be developed with the assurance level of the highest failure severity category associated with the monitored function.

SM.03—isolation between monitoring and monitored functions

The monitoring function shall not be affected by failures of the monitored function, i.e., the same failures it is supposed to monitor.
Note: This can be achieved, for instance, by assigning the monitoring and monitored functions to different partitions, which could be even allocated to different hardware.

SM.04—detection of timing failures

If a task timing violation occurs, the operating system shall be able to detect the violation and shall generate an event that shall be handled at the application level.
Requirement Rationale: If a timing violation occurs it means that the integrity of the system has been compromised; the decision on how to deal with it (e.g. activation of protective mechanism) is project specific and shall be handled at the application level.

Table 2 Traceability matrix

Req. ID	Req. title	Section/solution
TP.01	Time budget	8.1
TP.02	Inter-arrival rate	8.1
TP.03	Protection against timing failure propagation	8.1
TP.04	Time partitioning software development assurance level	8.1
SP.01	Logical isolation	Hardware support
SP.02	Hardware enforced protection	Hardware support
SP.03	Memory space definition	Hardware support
SR.01	Intra-partition shared resources	8.4
SR.02	Inter-partition shared resources	8.4
SR.03	Budget expiration before resource access	8.4
MO.01	Change of modes of operation	8.2
GD.01	Safe graceful degradation	8.2, 8.3
GD.02	Task suspension	8.2
SM.01	Fault analysis	8.5
SM.02	Monitoring function assurance level	8.5
SM.03	Isolation between monitoring and monitored functions	8.5
SM.04	Detection of timing failures	8.5

8 Potentially certifiable academic solutions and open problems

In Sect. 6 we focused our analysis on the academic solutions that are based on the theoretical MCS model originally based on the Vestal model, and have presented arguments demonstrating the limitations of those solutions in complying with the requirements of safety-related standards. Then in the previous section we have compiled a general set of desirable requirements for implementing MCS solutions with certification potential. Next we proceed to the identification of existing state-of-the-art solutions that are potentially good candidates to comply with those requirements, i.e., academic solutions that have good certification potential. We have also created a traceability (Table 2) between the MCS requirements and the respective academic solution addressing the requirement. The goal is to demonstrate that several existing state-of-the-art academic solutions can fulfil several of those requirements, despite the fact that combining or integrating those individual solutions into a complete MCS solution compliant with the standards may not be at all straightforward to achieve, and is still under investigation by both the industry and academy. It is also important to note that space partitioning is addressed by requirements SP.01 to SP.03. Due to the high level of trust that is required to perform this function (e.g. physical or virtual memory address protection), the only means to achieve the desired levels of reliability to ensure safety is through hardware support features, whose discussion is outside the scope of this work.

8.1 Resource reservation

8.1.1 Hierarchical scheduling

Resource reservation under hierarchical scheduling is another technique that lends itself very nicely for the scheduling of MCS, because it allows for strong time partitioning between applications. These techniques limits the effects of overruns in applications with variable computation times, where each application is assigned a fraction of the available resources, which is enough to satisfy their timing constraints. As explained in Lipari and Bini (2005), many techniques have been proposed for extending resource reservation to hierarchical scheduling. These techniques that allow reservation of processing resources (CPU time) build on the concept of server as the main schedulable entity. These techniques limits the effects of overruns in applications with variable computation times, where each application is assigned a fraction of the available resources, which is enough to satisfy their timing constraints. It is the real-time operating system responsibility to ensure temporal protection between applications, i.e. that each application does not consume more than the allocated amount of CPU. Therefore, an application which is allocated a fraction B_i of the total processor bandwidth behaves as if it were executing alone on a slower processor with a speed equal to B_i times the total processor speed.

A server is characterized by two tunable parameters (Q_i, P_i), where Q_i is the server maximum budget and P_i is the server replenishment period. The ratio $B_i = Q_i/P_i$ is known as the server bandwidth. According to the resource reservation technique, one or more tasks τ_i can be attached to a server S_i. Whenever the scheduler chooses to run S_i, one of the tasks assigned to S_i is executing and the budget is decreasing. Once the budget runs out the task cannot execute any more. The server's budget is replenished periodically every P_i time units.

The advantage of this method is that the isolation of each task is guaranteed at the CPU core level, i.e., the respect of their deadlines do not depend on the behaviour of other tasks associated to other servers as they always receive their reserved bandwidth. Therefore, hierarchical scheduling is a technique that naturally fulfils the time partitioning needs of MCS, which have been expressed in requirements TP.01 to TP.03 (see Table 2).

Although this method is very effective to achieve predictability in a platform where hard, soft and non-real time coexist, the overall system performance is very dependent on the correct computation and optimisation of the bandwidth allocation. For instance, if the CPU bandwidth allocated to a task is too small, then some tasks served by the server might not be able to respect all their deadlines. On the other hand, if the allocated bandwidth is larger than the actual task needs, the system may become underutilized, thus wasting valuable system resources. Therefore, a trade-off needs to be performed to ensure an acceptable level of performance, but still ensuring the safety properties of the system.

Several state-of-the-art solutions exist that implement resource reservation schemes for both single-core and multi-core platforms. Several of them are addressed in Burns and Davis (2013), including: Checconi et al. (2009), Davis and Burns (2005), and Lipari and Bini (2005).

8.1.2 Time-triggered scheduling

Currently, time-triggered (TT) scheduling is a common technique used in many safety-critical systems (e.g. aeronautic systems—see Sect. 5.1). In TT scheduling, the entire schedule is built off-line, during design time. During run-time, tasks are then activated at pre-defined instants of time. Under this approach, predictability and timing isolation are strictly ensured, thus easing the system certification process. The main challenge of this approach is the generation of a schedule that satisfies the requirements of all safety-critical applications that need to run on the system (Puffitsch et al. 2015).

8.2 Multi-mode scheduling

Multi-mode scheduling has similarities with the Vestal model, but it is actually more generic and hence has more potential to fulfil the MCS requirements specified in Sect. 7. Multi-mode scheduling focuses on real-time systems wherein the executing task set may undergo a structural change at runtime, which may include a modification of the task parameters, removing one or multiple tasks, and/or adding one or multiple tasks to the system. Such a reconfiguration may be imposed by a change in the physical environment, or requested to maintain system operation. For example in an avionics system, the tasks that must execute during taxing are different from those that must execute during flying.

Systems in which tasks may be updated and/or replaced for other tasks are organized in different “modes”. Each mode comprises a set of tasks and the entire system is thus called a “multi-mode” system. During the execution of such systems, switching from the current mode to any other mode requires to substitute the currently executing set of tasks with that of the new mode. This substitution introduces a transient phase during which tasks of both the old and new modes may run and be scheduled concurrently, thereby leading to a possible overload that can compromise the system schedulability, even if both the old and new modes have been asserted schedulable separately. When developing MCS, these multi-mode system models and their mode-switching protocols and analysis techniques could be used to preserve and guarantee the timing properties of all the tasks in the system while performing a mode change (to meet the requirement MO.01 described in the previous section). Moreover, the mode change solution is capable of ensuring that the tasks that have not been suspended (essential tasks) or re-activated (non-essential tasks) will still meet all their deadlines. Therefore, multi-mode scheduling is a good solution to achieve safe graceful degradation during the MCS operation, thus fulfilling also requirements GD.01 and GD.02.

In the past years, researchers have proposed multiple protocols to decide when a task in a new mode can be activated and start running (Ahmed et al. 2012; Junsung Kim et al. 2012; Goossens and Richard 2013; Nelis et al. 2011; Rattanatamrong and Fortes 2011; Phan et al. 2010; Hang and Hansson 2012; Lee and Shin 2013). Burns (Burns 2014), for instance, presents a comparison of Vestal's model with mode-change and how they relate to each other. Researchers have also proposed analysis methods that can prove, for a given scheduling algorithm in each mode and a given mode-change protocol, that the timing requirements of all the tasks will always be met. The scheduling problem during a transition between modes is manyfold and its complexity depends on the behaviour and requirements of the tasks in the old and new mode when a mode change is initiated.

8.3 Elastic task model

To fit with industrial requirements, several research works focus on the elastic tasks model instead of the discarding tasks model (Buttazzo et al. 1998, 2002; Buttazzo and Abeni 2002). Under this framework, periodic tasks can change their execution time based on the system load. System overload conditions can be handled in a more flexible manner, by degrading the quality of service through the adaptation of the tasks' periods.

8.4 Resource sharing protocols

As previously explained, an MCS implementation that complies with the safety-related standards must allow independent development teams (e.g. subcontractors) to produce software applications that are further integrated to form a complete final product. To achieve this goal, it is desirable that the timing behaviour of the several subsystems be independent, even if they share mutually exclusive resources (e.g. semaphores or mutexes). This need is reflected in requirements SR.01 to SR.03, which can be effectively addressed by state-of-the-art resource sharing protocols (see mapping in Table 2), which are briefly discussed hereafter.

Resource control policies for single-core processors are well established and developed (Sha et al. 1990; Audsley et al. 1995; Baker 1991). These protocols are compliant with the standards and most of them are either partially or fully supported by the existing industrial solutions and therefore this is not an open problem any more. Most of the existing solutions for resource sharing in multi-core are extensions of single-core approaches (Rajkumar et al. 1988; Rajkumar 1990; Chen and Tripathi 1994; López et al. 2004; Gai et al. 2001; Lakshmanan et al. 2009; Easwaran and Andersson 2009). Fewer solutions were conceived specifically for multi-core (Devi et al. 2006; Block et al. 2007).

When hierarchical scheduling is involved, the problem becomes more complex because it must be ensured that one task in a server S_a that shares resources with another task in server S_b does not suffer from unbounded blocking when trying to access the resource. Several solutions exist that implement resource sharing with time partitioning such as Faggioli et al. (2010), Inam et al. (2014), Biondi et al. (2013), Bertogna et al. (2009), and Davis and Burns (2006). The solution proposed in Brandenburg (2014) has been specifically designed for MCS.

When combining resource sharing protocols with servers, one of the most important problem that needs to be addressed is related to the exhaustion of a server's budget when one of its served tasks is executing a critical section (i.e. an inter-partition shared resource). Several approaches to solve this problem have been proposed. The first solution is to allow the server to consume some extra budget until the served task finalizes the execution of the critical section (Abeni and Buttazzo 2004; Davis and Burns 2006; Behnam et al. 2008, 2010). However, allowing such extra bandwidth requirement violates the servers' temporal isolation property. Another solution to this problem was proposed in the SIRAP protocol (Behnam et al. 2007). SIRAP introduces a budget check before access to the shared resource is granted, i.e. if the servers budget is not sufficient for one of its executing tasks to complete the execution of a critical section, the access to the resource is only allowed after the next budget replenishment. The disadvantage of this approach is that it penalizes the response time of the served tasks. A third solution to this problem was proposed by the BROE (Bounded-Delay Resource Open Environment) approach (Bertogna et al. 2009). Under BROE, when a task attempts to access a critical section and the budget is not sufficient, the full budget is replenished at the earliest possible time, so that both the server and the maximum task response time is preserved. More recently, the synchronous mixed-criticality IPC protocol(MC-IPC) (Brandenburg 2014) has been proposed. MC-IPC servers use the concept of *bandwidth inheritance*: with that solution, one server may undertake the processing of a resource critical section on behalf of another task assigned to another server.

Therefore, the problem of sharing resources within a time partition (see requirement SR.01) is reduced to a single-core problem and can effectively be addressed by the consolidated single-core protocols previously referred. When considering resources sharing between partitions (see requirements SR.02 and SR.03), the problem becomes more complex and although several solutions exist, only MC-IPC (Brandenburg 2014) effectively addresses MCS by ensuring logical and temporal isolation of tasks of different criticalities. However, even though this solution is quite promising and has a series of advantages, it still needs to be tested and validated in the field with real systems (e.g. avionics or automotive) and then undergo evaluation by certification authorities. In this process, some disadvantages are foreseen, such as the high migration cost of the server that encapsulates the resources shared between partitions (due to bandwidth inheritance) and the overall complexity of the solution, which will make the construction of the safety argument not straightforward to achieve, in order to successfully convince the certification authorities. Hence, a solution that combines resource sharing with servers still remains an important research direction in the context of MCS.

8.5 Runtime verification

In the past twenty years, Runtime Verification (RV) has been growing considerably and became widely recognized as a discipline (Leucker and Schallhart 2009), in the broader area of formal verification. It has a special place as a complement for the more traditional static approaches, such as Model Checking (Baier and Katoen 2008) and Theorem Proving (Bertot and Castéran 2004). RV can be seen as a side product of Model Checking with Runtime Monitoring as the supporting technology. RV consists in enriching a target system with monitors capable of observing the system behaviour by means of the analysis of finite traces of events resulting from its execution. Differently from runtime monitoring, RV verifies that the requirements of the system and/or some of its components are respected at runtime. Those requirements are defined using well-formed and semantically sound specifications written in a formal language of choice, e.g., Linear Temporal Logic, Regular Expressions, etc. The monitors can then be automatically generated using formally correct methods, which ensure that whatever may cause the requirement to not be observed, the anomaly will always be detected. The monitors can be used to detect and sometimes predict errors, faults, failures or simply abnormal deviations from the expected system behaviour. That information can finally be used to provide compensating provisions (and thus satisfying requirement SM.04), which may for instance implement graceful degradation, trigger a mode change, deactivate or isolate the faulty component.

RT-MaC (Sammapun et al. 2005) is a good example of a complete runtime verification solution. The RT-MaC system is a real-time extension of its precursor MaC (Kim et al. 2002). RT-MaC allows to define functional requirements, time-bound conditions and probabilistic properties to be verified at runtime. Many other RV frameworks are described in the state-of-the-art, some of which became commercially available. Notorious examples are Eagle (d'Amorim and Havelund 2005), Time Rover,[9] MOP (Chen and Roşu 2007), RMOR (Havelund 2008), RuleR (Barringer et al. 2009) and RV-Monitor (Luo et al. 2014). MOP and RuleR for instance have the flexibility to allow users to define RV specifications using different input formal languages, which makes these platforms very flexible depending on the target system they intend to monitor. Their major drawback though is their intrusiveness. Most of those frameworks add calls to verification procedures into the application code. This modification of the target application may be problematic in safety critical systems if it is not planned from the very beginning of the system design. Furthermore, they do not ensure isolation between the monitors and the monitored application. Hence, a failure of the monitored application may very well impact the capability of the monitor to detect that failure.

The isolation problem for runtime verification in MCS has been recently studied in Nelissen et al. (2015). A runtime monitoring architecture was proposed as a variation of Chodrow et al. (1991). The presented solution allows for a complete time and space isolation between monitors and the monitored applications if implemented on top of a hierarchical scheduling framework, hence complying with requirement SM.03.

Therefore, if runtime verification technique is integrated early in the system design process, it can be a good candidate software solution (thus covering SM.03) for verifying if the timing requirements of a MCS are satisfied during runtime (thus covering requirement SM.04). It was also previously discussed that runtime verification can also provide good independence between monitoring and monitored application if implemented on top of a hierarchical scheduling framework, as required by SM.03. Requirements SM.01 and SM.02 are related to the development assurance process, but the activities therein specified still need to be performed for achieving a solution that is compliant with the safety related standards.

[9] http://www.time-rover.com.

8.6 Assignment of a software failure rate

Recent research on the mixed-criticality scheduling theory have introduced the concept of probabilistic WCET (Davis et al. 2014), which can be understood as the "probability of violating a timing requirement". In this model, from a mathematical point of view the WCET of a task is no longer a single rigid value. Rather, the model provides a threshold, i.e., an upper-bound on the execution time, that has a given probability of being exceeded at runtime. Informally, the rational behind the introduction of probabilities in the model is the assumption that if the probability of exceeding that threshold is shown to be smaller than the probability of experiencing an irreversible failure (like an irreversible hardware failure for instance), then that threshold can be used as a "safe" estimate of the WCET. Recent papers present various techniques to derive such probabilistic WCET (Abella et al. 2014). In those papers the target probabilities are either taken directly from the probabilities of failure of a safety function allocated to the E/E/PE safety-related system (like in Davis et al. (2014)), e.g., probability of failure of 10^{-9} per hour for a SIL 4 function as defined in Table 3 of IEC61508-1 (or in the FAA Advisory Circular AC-25-1309), or for the same reasons they are set to even lower values (Abella et al. 2014).

These probabilistic techniques aim at building a reliability model of the software, according to which confidence can be placed in the expected timing behavior of the application and in particular in its worst-case responsiveness. In broad terms, *software* reliability is the property of the software being "free from faults". Failures caused by software can degrade the system performance, up to the complete loss of the system or potential loss of life or major damage to the environment. According to this definition, exceeding a pre-allocated execution budget can indeed be seen as a software fault as it may bring about the same consequences. There are, however, three important points that we would like to discuss in this paper.

First, although the rationale behind this probabilistic model is well motivated and fully justified (from the community point of view), in most of recent research works on probabilistic timing estimates the research community has shown a particularly high confidence in the applicability of their model to real systems. In some of those papers, sometimes it seems granted that this probabilistic model will soon be used in MC system V&V processes. However, it must be highlighted that at the time of writing this paper, although numerous consolidated reliability models have been developed to *quantitatively* assess the compliance of the *hardware* components to the reliability requirements (RIAC-HDBK-217Plus 2006), *software* reliability models are still under debate within industry, academia, and international standards community. Those models rely on a number of assumptions that have proven not to be fully justified in the vast majority of bespoke software and currently in the industry, confidence cannot be placed in such models to assess the reliability of the software parts. This is clearly and explicitly stated, for instance, in Sect. 4.1.2 of ECSS-Q-HB-80-03A (2009) or Sect. 12.3.3 of the DO-178C:

"Many methods for predicting software reliability based on developmental metrics have been published, for example, software structure, defect detection rate, etc. This document does not provide guidance for those types of methods, because at the time of writing [2011], currently available methods did not provide results in which confidence can be placed."—Sect. 12.3.3, p. 89 of DO-178C (DO-178C 2011).

Second, these probability thresholds (e.g. 10^{-5}, 10^{-9}, etc.) have been defined *at the system level* as function failure rates and to the best of our knowledge, there is no publications whatsoever that discuss why those specific thresholds could be applied to the software parts of the system. Last but not least, until now the safety and dependability assessment of safety-critical *software* has always been performed through a set of *qualitative* processes that are applied during all phases of the software development life-cycle. To the best of our knowledge, there is currently no evidence indicating that the process of assessing *software* safety is about to change from a *qualitative* process to a *quantitative* process. Unlike the typical process applied to develop the hardware, the development of a software to a certain assurance level does not imply the assignment of a failure rate for that software. In other words, there is no evidence that those specific thresholds (or any other thresholds) applied to function failure rates will ever apply in the assessment of software safety.

Although the previous arguments demonstrate the limitations of the probabilistic WCET software models to justify safety aspects, it is again important to highlight (similarly to the Vestal's model) that the objective of this work is not to question the usefulness or credibility of those methods, but rather to demonstrate that in the safety-critical domain it will be extremely difficult to prove to the certification authorities that, by using these approaches, the safety of the system will be ensured even under the most unfavourable conditions. However, the probabilistic WCET estimation can be considered as a very important and useful tool to support the design of mixed-criticality systems, mainly in terms of determining more realistic estimates of the tasks WCETs. Instead of the very conservative values that are typically used to define the safety margins to be applied in the actual system design, this technique has the potential to allow the determination of much more accurate WCET estimations, hence enhancing the cost-effectiveness of the system design.

Therefore, at this date, even though the computation of probabilistic estimates to MCS constitutes an important research direction that aims at improving the WCET estimation of real-time tasks, it cannot be taken as granted that those estimates will ever be used in industrial systems to prove the safety of a software function.

8.7 Open problems

After the identification of the state-of-the-art solutions that address the proposed MCS requirements, we now perform an evaluation of the integration potential of some of those solutions in the same MCS platform, i.e. whether those solutions are compatible with each other. Note that not all possible combinations of techniques herein presented were analysed. We have only evaluated those solutions whose integration seem pertinent as a research direction.

As discussed in the previous subsections, hierarchical scheduling and resource sharing are two techniques that can be integrated in the same MCS model. In a time partitioned environment, the need of a resource sharing protocol that allows tasks in different partitions to synchronize the access to a shared resource in a predictable way is evident. As discussed in Sect. 8.4, several state-of-the-art solutions exist that have demonstrated the feasibility of combining these two techniques in the same platform. However, as also briefly discussed in Sect. 8.4, the integration of these two techniques poses several challenges, such as the well known problem of server budget exhaustion inside a critical section. Therefore, even though several sophisticated solutions exist for resource sharing in a hierarchical scheduling approach, there is still margin for improvements, especially in the constrained context of mixed-criticality systems.

The use of runtime verification on top of a hierarchical scheduling framework was also addressed in Sect. 8.4. These two techniques provide a nice integration, where the timing isolation provided by the partitions improve the reliability of the monitors, by guaranteeing their isolation and independence from the monitored applications, thus enforcing requirement SM.03.

Considering the techniques presented, two fundamental problems remain open, both of them related to multi-mode change solution. The first is related to the integration between multi-mode change and hierarchical scheduling. In order to integrate these two solutions, the schedulability of the system would need to be ensured at all levels during mode transition, i.e., at the level of the top scheduler, responsible to schedule the servers (time partitions), and at the level of the local scheduler, responsible to schedule the tasks within the server. To the best of our knowledge, this still remains an open problem.

The second problem refers to the integration between multi-mode change and resource sharing protocols. A solution potentially integrating these two techniques would need to ensure an upper bound for the time to access a shared resource even during a transition between the old and new modes. To the best of our knowledge, this still also remains an open problem.

9 Conclusion

In this paper we discussed the mixed-criticality systems (MCS) concept from an industrial perspective. The main objective was to ease the understanding of the MCS challenges and constraints for non-experts, and hence stimulate the development of adequate solutions while avoiding the propagation of inaccurate assumptions.

To achieve that goal, we started by presenting an overview of the general process for the development of safety-critical systems in compliance with the relevant safety-related standards from several industrial domains (avionics, space, industrial process, automotive). The concepts presented in this introductory part are essential for the accurate understanding of the concept of *criticality* and its allocation to subsystems and components.

After the introduction of the MCS concept, we performed a review of the most important architectural considerations and requirements from three safety-related industrial standards, with respect to the development of MCS. Some of these techniques are well known to the real-time community, but some of them are especially relevant for the development of mixed-criticality systems, such as partitioning and graceful degradation.

We concluded the “industry-oriented” part of the paper by presenting the industrial solutions that prevail in the aeronautic and automotive application domains (ARINC-653 and AUTOSAR, respectively) to design mixed-criticality systems in accordance with their domain-specific requirements. The design and development of industrial MCS in accordance with such solutions greatly facilitate the certification process, because they specify architectural and design requirements that are compliant with the stringent requirements of the safety-related standards, especially with respect to the isolation and independence of the MCS applications.

We then moved to the “academy-oriented” part of the paper, where we discussed the strengths and weaknesses of the theoretical model of MCS found in the academic literature. Based on references and examples from the safety-related industrial standards, we highlighted some misalignments that exist in the interpretation of some key concepts used in the MCS scientific literature and those standards, especially in what concerns the notion of software task assurance level and the notion of importance.

As a preliminary step towards a MCS model that is compliant with the safety-related standards, we presented some key requirements that must be captured and taken into consideration in the model. And finally, we identified and discussed some academic state-of-the-art solutions that are potentially compliant with those requirements.

It is rather difficult to discuss about potential trends and perspectives regarding the adoption of MCS by the industry, mainly because of the considerable amount of open problems that currently still exist. To perform this evaluation, we believe the most adequate approach is by evaluating some of the industrial sectors (or domains) separately. The avionic sector is known to be very conservative, and currently there is not indication that this approach is going to change. TT scheduling will continue to be the main technique to be used during the coming years. Moreover, with the advent of multi-core, this sector is being faced with several additional challenges, such as ensuring predictability with resource sharing in those platforms. The automotive sector is likely to become the sector where MCS will face the fastest development, mainly because there is no central certification authority, like in the aeronautic sector, where entities like FAA (Federal Aviation Administration) or EASA (European Aviation Safety Agency) perform a very strong technical and safety regulation. The ISO26262 is a more recent standard in relation to other industrial standards, and is an important step toward the safer development of automobiles, whose amount of software implemented functionalities (critical and non-critical) is facing an incredible growth during recent years. In relation the more traditional industrial sector (e.g. oil and gas), the need for concurrently running critical and non-critical applications in the same platform will be more limited (because of no direct interaction with individual consumers), but at some point the MCS approach may bring significant benefits in terms of cost reduction. It is also our belief that the MCS can be useful for the development of mission-critical (e.g. military or space systems), where compliance with dependability requirements become the main concern. MCS solutions will also become interesting for the railway sector, although the development may not be so fast as in the automotive sector, because it is a more conservative industry. Nevertheless, the amount of critical and non-critical functionalities is increasingly growing (especially the latter) and currently processing platforms are facing great challenges in terms of performance, with safe and non-safe single-core embedded computers are running near their performance limits.

Acknowledgements This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and when applicable, co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within Project UID/CEC/04234/2013 (CISTER Research Centre); also by, FCT/MEC and the EU ARTEMIS JU within projects ARTEMIS/0003/2012—JU Grant nr. 333053 (CONCERTO) and ARTEMIS/0001/2013—JU Grant nr. 621429(EMC2); and also by the Portuguese National Innovation Agency (ANI) under the ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project V-SIS, QREN—SI I&DT nr. 38923.

References

Abella J, Hardy D, Puaut I, Quinones E, Cazorla F (2014) On the comparison of deterministic and probabilistic wcet estimation techniques. In: ECRTS, pp 266–275

Abeni L, Buttazzo G (2004) Resource reservation in dynamic real-time systems. Real Time Syst 27(2):123–167

Ahmed M, Fisher N, Grosu D (2012) A parallel algorithm for edf-schedulability analysis of multi-modal real-time systems. In: IEEE 18th international conference on embedded and real-time computing systems and applications (RTCSA), pp 154–163. https://doi.org/10.1109/RTCSA.2012.49

ARINC 700 series: Arinc (2015) http://store.aviation-ia.com/cf/store/catalog.cfm?prod_group_id=1&category_group_id=4

ARP4761 (1996) Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. SAE International, Warrendale

ARP4761A (1996) Guidelines for development of civil aircraft and systems. SAE International, Warrendale

Audsley NC, Burns A, Davis RI, Tindell KW, Wellings AJ (1995) Fixed priority pre-emptive scheduling: an historical perspective. Real Time Syst 8(2–3):173–198

AUTOSAR (2011) Technical Overview, V2.2.2 R3.2 Rev 1. AUTOSAR

AUTOSAR (2013) Requirements on operating system, V3.1.1 R4.1 Rev 2. AUTOSAR

AUTOSAR (2015) Complex driver design and integration guideline, R4.2.2. AUTOSAR

Baier C, Katoen JP (2008) Principles of model checking, vol 26202649. MIT Press, Cambridge

Baker TP (1991) Stack-based scheduling of realtime processes. Real Time Syst 3(1):67–99

Barringer H, Havelund K, Rydeheard D, Groce A (2009) Rule systems for runtime verification: a short tutorial. In: Runtime Verification, pp. 1–24. Springer, New York

Baruah SK, Burns A, Davis RI (2011) Response-time analysis for mixed criticality systems. In: IEEE 32nd Real-time systems symposium (RTSS), pp. 34–43

Behnam M, Shin I, Nolte T, Nolin M (2007) Sirap: a synchronization protocol for hierarchical resource sharingin real-time open systems. In: Proceedings of the 7th ACM & IEEE international conference on Embedded software, pp 279–288. ACM

Behnam M, Shin I, Nolte T, Nolin M (2008) Scheduling of semi-independent real-time components: Overrun methods and resource holding times. In: IEEE international conference on emerging technologies and factory automation (ETFA 2008), pp 575–582

Behnam M, Nolte T, Sjödin M, Shin I (2010) Overrun methods and resource holding times for hierarchical scheduling of semi-independent real-time systems. IEEE Trans Ind Inform 6(1):93–104

Bertogna M, Fisher N, Baruah S (2009) Resource-sharing servers for open environments. IEEE Trans Ind Inform 5(3):202–219

Bertot Y, Castéran P (2004) Interactive theorem proving and program development: CoqArt: the calculus of inductive constructions. Springer, New York

Biondi A, Buttazzo G, Bertogna M (2013) Schedulability analysis of hierarchical real-time systems under shared resources. Technical Report TR-13-01

Block A, Leontyev H, Brandenburg BB, Anderson JH (2007) A flexible real-time locking protocol for multiprocessors. In: 13th IEEE international conference on embedded and real-time computing systems and applications (RTCSA 2007), pp 47–56

Brandenburg BB (2014) A synchronous ipc protocol for predictable access to shared resources in mixed-criticality systems. In IEEE real-time systems symposium (RTSS), pp 196–206

Burns A (2014) System mode changes-general and criticality-based. In: Proceedings of 2nd workshop on mixed criticality systems (WMC), pp 3–8

Burns A, Davis R (2013) Mixed criticality systems-a review. Department of Computer Science, University of York, Tech. Rep

Buttazzo GC, Lipari G, Abeni L (1998) Elastic task model for adaptive rate control. In: IEEE proceedings of the 19th real-time systems symposium, pp 286–295

Buttazzo G, Abeni L (2002) Adaptive workload management through elastic scheduling. Real Time Syst 23(1):7–24

Buttazzo GC, Lipari G, Caccamo M, Abeni L (2002) Elastic scheduling for flexible workload management. IEEE Trans Comput 51(3):289–302

Checconi F, Cucinotta T, Faggioli D, Lipari G (2009) Hierarchical multiprocessor cpu reservations for the linux kernel. In: Proceedings of the 5th international workshop on operating systems platforms for embedded real-time applications (OSPERT 2009), Dublin, Ireland, pp 15–22

Chen F, Roşu G (2007) Mop: an efficient and generic runtime verification framework. In: ACM SIGPLAN Notices, vol. 42, pp. 569–588. ACM

Chen CM, Tripathi SK (1994) Multiprocessor priority ceiling based protocols. Tech. rep, College Park, MD, USA

Chisholm M, Ward BC, Kim N, Anderson JH (2015) Cache sharing and isolation tradeoffs in multicore mixed-criticality systems. In: IEEE real-time systems symposium, pp 305–316

Chisholm M, Kim N, Ward BC, Otterness N, Anderson JH, Smith FD (2016) Reconciling the tension between hardware isolation and data sharing in mixed-criticality, multicore systems. In: IEEE real-time systems symposium (RTSS), pp 57–68

Chodrow SE, Jahanian F, Donner M (1991) Run-time monitoring of real-time systems. In: IEEE proceedings of the 12th real-time systems symposium (RTSS 1991), pp 74–83

d'Amorim M, Havelund K (2005) Event-based runtime verification of java programs. In: ACM SIGSOFT software engineering notes, vol. 30, pp. 1–7. ACM

Davis RI, Burns A (2005) Hierarchical fixed priority pre-emptive scheduling. In: 26th IEEE international real-time systems symposium (RTSS), pp 10–pp

Davis RI, Burns A (2006) Resource sharing in hierarchical fixed priority pre-emptive systems. In: 27th IEEE international real-time systems symposium (RTSS'06), pp 257–270

Davis R, Vardanega T, Alexanderson J, Francis V, Mark P, Ian B, Mikel AA, Wartel F, Cucu-Grosjean L, Mathieu P, Glenn F, Cazorla FJ (2014) PROXIMA: a probabilistic approach to the timing behaviour of mixed-criticality systems. Ada User J 2:118–122

Devi UC, Leontyev H, Anderson JH (2006) Efficient synchronization under global EDF scheduling on multiprocessors. In: IEEE 18th Euromicro conference on real-time systems, pp 10

Diniz N, Rufino J (2005) Arinc 653 in space dasia 2005, eurospace, edinburgh, scotland

DO-178C (2011) Software considerations in airborne systems and equipment certification. RTCA, Inc

Easwaran A, Andersson B (2009) Resource sharing in global fixed-priority preemptive multiprocessor scheduling. In: 30th IEEE real-time systems symposium (RTSS 2009), pp 377–386

ECSS-Q-HB-80-03A (2009) Space product assurance—software dependability and safety. European Cooperation for Space Standardization

ECSS-Q-ST-40C (2009) Space product assurance—dependability. European Cooperation for Space Standardization

ECSS-Q-ST-40C (2009) Space product assurance–safety. European Cooperation for Space Standardization

ECSS-Q-ST-80C (2009) Software product assurance. European Cooperation for Space Standardization

EN 50128 (2009) Railway applications communication, signalling and processing systems software for railway control and protection systems. CENELEC

Ernst R, Di Natale M (2016) Mixed criticality systems-a history of misconceptions? IEEE Des Test 33(5):65–74

Faggioli D, Lipari G, Cucinotta T (2010) The multiprocessor bandwidth inheritance protocol. In: IEEE 22nd Euromicro conference on real-time systems (ECRTS), pp. 90–99

Gai P, Lipari G, Di Natale M (2001) Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In: 22nd IEEE proceedings real-time systems symposium (RTSS), pp 73–83

Goossens J, Richard P (2013) Partitioned scheduling of multimode multiprocessor real-time systems with temporal isolation. In: Proceedings of the 21st international conference on real-time networks and systems (RTNS ’13), pp 297–305. ACM, New York. https://doi.org/10.1145/2516821.2516822

Hang Y, Hansson H (2012) Timing analysis for mode switch in component-based multi-mode systems. In: 24th Euromicro conference on real-time systems (ECRTS), pp. 255–264. https://doi.org/10.1109/ECRTS.2012.23

Havelund K (2008) Runtime verification of C programs. Springer, New York

IEC61508 (2010) Functional safety of electrical/electronic/programmable electronic safety-related systems. IEC

Inam R, Mahmud N, Behnam M, Nolte T, Sjödin M (2014) The multi-resource server for predictable execution on multi-core platforms. In: IEEE 20th real-time and embedded technology and applications symposium (RTAS), pp 1–10

ISO26262 (2011) Road vehicles—functional safety. ISO

Kim M, Lee I, Sammapun U, Shin J, Sokolsky O (2002) Monitoring, checking, and steering of real-time systems. Electron Notes Theor Comput Sci 70(4):95–111

Kim J, Lakshmanan K, Rajkumar RR (2012) Rhythmic tasks: a new task model with continually varying periods for cyber-physical systems. In: The 2012 IEEE/ACM third international conference on cyber-physical systems, pp 55–64

Kim N, Ward BC, Chisholm M, Fu CY, Anderson JH, Smith FD (2016) Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning. In: IEEE real-time and embedded technology and applications symposium (RTAS), pp 1–12

Lakshmanan K, de Niz D, Rajkumar R (2009) Coordinated task scheduling, allocation and synchronization on multiprocessors. In: 30th IEEE real-time systems symposium (RTSS 2009), pp 469–478

Lee J, Shin KG (2013) Schedulability analysis for a mode transition in real-time multi-core systems. In: IEEE 34th real-time systems symposium (RTSS), pp 11–20. https://doi.org/10.1109/RTSS.2013.10

Leucker M, Schallhart C (2009) A brief account of runtime verification. J Logic Algebr Program 78(5):293–303

Lipari G, Bini E (2005) A methodology for designing hierarchical scheduling systems. J Embed Comput 1(2):257–269

López JM, Díaz JL, García DF (2004) Utilization bounds for edf scheduling on real-time multiprocessor systems. Real Time Syst 28(1):39–68

Luo Q, Zhang Y, Lee C, Jin D, Meredith PO, Şerbănuţă TF, Roşu G (2014) Rv-monitor: efficient parametric runtime verification with simultaneous properties. In: Runtime verification, pp 285–300. Springer, New York

Nelis V, Andersson B, Marinho J, Petters SM (2011) Global-edf scheduling of multimode real-time systems considering mode independent tasks. In: 23rd Euromicro conference on real-time systems (ECRTS), pp 205–214

Nelissen G, Pereira D, Pinho LM (2015) A novel run-time monitoring architecture for safe and efficient inline monitoring. In: Reliable software technologies–Ada-Europe 2015, pp 66–82. Springer, New York

Phan LTX, Lee I, Sokolsky O (2010) Compositional analysis of multi-mode systems. In: 22nd Euromicro conference on real-time systems (ECRTS), pp 197–206. https://doi.org/10.1109/ECRTS.2010.35

Puffitsch W, Noulard E, Pagetti C (2015) Off-line mapping of multi-rate dependent task sets to many-core platforms. Real Time Syst 51(5):526–565

Rajkumar R (1990) Real-time synchronization protocols for shared memory multiprocessors. In: IEEE 10th international conference on proceedings distributed computing systems, pp 116–123

Rajkumar R, Sha L, Lehoczky JP (1988) Real-time synchronization protocols for multiprocessors. In: RTSS, pp 259–269

Rattanatamrong P, Fortes JAB (2011) Mode transition for online scheduling of adaptive real-time systems on multiprocessors. In: IEEE 17th international conference on embedded and real-time computing systems and applications (RTCSA), vol 1, pp 25–32. https://doi.org/10.1109/RTCSA.2011.71

RIAC-HDBK-217Plus (2006) Handbook of 217Plus reliability prediction models. RIAC

Sammapun U, Lee I, Sokolsky O (2005) Rt-mac: Runtime monitoring and checking of quantitative and probabilistic properties. In: 11th IEEE international conference on embedded and real-time computing systems and applications, pp 147–153

Santy F, Raravi G, Nelissen G, Nelis V, Kumar P, Goossens J, Tovar E (2013) Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems. In: RTNS, pp 183–192. ACM

Sha L, Rajkumar R, Lehoczky JP (1990) Priority inheritance protocols: an approach to real-time synchronization. IEEE Trans Comput 39(9):1175–1185

Vestal S (2007) Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In: IEEE RTSS, pp 239–243

Watkins CB, Walter R (2007) Transitioning from federated avionics architectures to integrated modular avionics. In: IEEE DASC'07, pp 2-A

Alexandre Esper is an experienced systems and software engineer. He holds a degree (1998) in Electrical Engineering from the State University of Campinas, Brazil, and a Master of Space Systems Engineering (2011) from the Delft University of Technology, in the Netherlands. He has been working as a senior engineer and technical manager at Critical Software S.A. since 2005, responsible to lead and execute projects of safety-critical systems and software for the aerospace, defence and transportation sectors, in compliance with international safety-related industrial standards. Before joining CRITICAL Software, he worked as a mobile communications systems engineer for NEC from 1998 to 2005, including projects in NEC Brazil, NEC UK and NEC Portugal. Currently he is also engaged in his PhD studies in Embedded and Real-Time Systems at the Faculty of Engineering of the University of Porto, in association with the Research Centre in Real-Time Computing Systems of the Polytechnic Institute of Porto.

Geoffrey Nelissen is a research scientist at CISTER (Research Centre in Real-Time and Embedded Computing Systems), a research centre co-hosted by the Faculty of Engineering of the University of Porto (FEUP) and the School of Engineering (ISEP) of the Polytechnic Institute of Porto. Prior to joining CISTER, he studied in Brussels at the Université Libre de Bruxelles (ULB), where he earned his PhD in January 2013 and his master degree in electrical engineering in 2008. His research activities are mostly related to the modelling and analysis of real-time and safety critical embedded systems. His research interests span all theoretical and practical aspects of real-time embedded systems design with an emphasis on the analysis and configuration of real-time applications on multicore and distributed platforms.

Vincent Nélis received his PhD degree in 2010 at the Computer Science Department of the Université Libre de Bruxelles, Belgium. Since then, Vincent has been working at the CISTER Research Center of Porto, Portugal. Throughout his career, he has published 65+ articles in international journals, conferences, and workshops, and has received 7 international awards for his work. His research work has been focused mainly on developing resource allocation techniques (mapping, scheduling, partitioning, and sharing algorithms) for embedded real-time systems and guaranteeing their expected temporal behavior through extensive simulations and timing analyses.

Eduardo Tovar has received the Licentiate, MSc and PhD degrees in electrical and computer engineering from the University of Porto, Porto, Portugal, in 1990, 1995 and 1999, respectively. Currently he his Professor in the Computer Engineering Department at the School of Engineering (ISEP) of Polytechnic Institute of Porto (IPP), where he is also engaged in research on real-time distributed systems, wireless sensor networks, multiprocessor systems, cyber-physical systems and industrial communication systems. He heads the CISTER Research Unit, an internationally renowned research centre focusing on RTD in real-time and embedded computing systems. He is deeply engaged in research on real-time distributed systems, multiprocessor systems, cyber-physical systems and industrial communication systems. He is currently the Vice-chair of ACM SIGBED (ACM Special Interest Group on Embedded Computing Systems) and was for 5 years, until December 2015, member of the Executive Committee of the IEEE Technical Committee on Real-Time Systems (TC-RTS). Since 1991 he authored or co-authored more than 150 scientific and technical papers in the area of real-time and embedded computing systems, with emphasis on multiprocessor systems and distributed embedded systems. He has been consistently participating in top-rated scientific events as member of the Program Committee, as Program Chair or as General Chair. Notably he has been program chair/co-chair for ECRTS 2005, IEEE RTCSA 2010, IEEE RTAS 2013 or IEEE RTCSA 2016, all in the area of real-time computing systems. He has also been program chair/co-chair of other key scientific events in the area of architectures for computing systems and cyber-physical systems as is the case of ARCS 2014 or the ACM/IEEE ICCPS 2016 or in the area of industrial communications (IEEE WFCS 2014). He is General Co-Chair of the CPSWeek 2018.