The Journal of Systems and Software 156 (2019) 204–216

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

New Trends and Ideas

Towards mixed criticality task scheduling in cyber physical systems: Challenges and perspectives

Eugenia Ana Capota[a], Cristina Sorina Stangaciu[a], Mihai Victor Micea[a], Daniel-Ioan Curiac[b,*]

[a] Computer and Information Technology Department, Politehnica University, Vasile Parvan 2, 300223, Timisoara, Romania
[b] Automation and Applied Information Department, Politehnica University, Vasile Parvan 2, 300223, Timisoara, Romania

ARTICLE INFO

Article history:
Received 13 January 2019
Revised 3 June 2019
Accepted 27 June 2019
Available online 27 June 2019

Keywords:
Cyber physical systems
Real-time scheduling
Mixed criticality systems
Multiple processing units

ABSTRACT

Cyber physical systems (CPSs) are a fast-evolving technology based on a strong synergy between heterogeneous sensing, networking, computation and control modules. When coping with critical applications that require real-time performance and autonomous operation in uncertain conditions, the design of such complex systems is still facing significant difficulties. A particular challenge in this respect derives from the software intensive nature of these systems - the need to develop flexible and specifically tailored task scheduling techniques. In our view, an appropriate line of thinking is to take advantage of mixed criticality concepts following the lessons learned from avionics and automotive domains, where complexity, safety, determinism and real-time constraints are extreme. From this perspective, our work aims at facilitating the integration of mixed criticality systems-based strategy in cyber physical systems by identifying the particularities of the latter and their influence on scheduling mechanisms, by describing the standard mixed-criticality task model in the cyber physical systems context, and by analyzing and proposing the most suitable scheduling algorithms to be implemented in cyber physical systems. Moreover, the perspectives on future developments in this area are discussed, as new horizons in research arise with the integration of mixed criticality concepts in the cyber physical systems context.

1. Introduction

Integrated circuits have rapidly evolved in terms of performance, while reducing their size, cost and power consumption. This evolution has led to an accelerated spreading of embedded systems in all human activity fields: ranging from military to domestic applications. Nevertheless, in special fields like automotive and avionics, where complexity, safety, determinism and real-time constraints are extreme, or in fields where hardware constraints are not to be neglected, there still is a stringent need for adequate scheduling mechanisms tailored to the specific needs of these applications.

Cyber physical systems (CPSs) are special systems, with an explosive development in the last years. These systems include a *logic function in their design such that either the state of the logic function could be altered by a change in the state of the real world, or the state of the real world could be modified by a change in the state of the logic function, or both* (Laplante, 2004). Basically, they include systems with different performances, hardware and time constraints which have a direct interaction with the environment, like control systems, robotic systems, transportation systems etc. Each requirement comes from different types of applications and thus, various kinds of scheduling mechanisms may be needed.

Having a flexible scheduling policy which can be tailored according to diverse requirements can represent a desideratum in numerous cases. Dealing with these complex challenges has led to the development of a new concept and a new class of systems: the Mixed Criticality Systems (MCSs). The MCS concept basically refers to "*an embedded computing platform in which application functions of different criticality share computation and/or communication resources*" (Ernst and Di Natale, 2016).

Even though MCSs originate from automotive and avionics areas, their applicability extends these domains, such systems being present also in fields like the Internet of Things (Kamienski et al., 2017; Carpenter et al., 2017), medical devices (Carpenter et al., 2017), industrial systems (Xia et al., 2017) and cyber physical systems in general.

The use of MCSs within CPSs requires a special kind of approach, because of the different requirements, which may seem

* Corresponding author.
E-mail addresses: eugenia.capota@cs.upt.ro (E.A. Capota), cristina.stangaciu@cs.upt.ro (C.S. Stangaciu), mihai.micea@cs.upt.ro (M.V. Micea), daniel.curiac@aut.upt.ro (D.-I. Curiac).

https://doi.org/10.1016/j.jss.2019.06.099
0164-1212/

contradictory sometimes. In many cases, the safety and real-time requirements must be matched with the restricted resources, low power and low cost constraints, which increase the level of complexity in the CPS case.

The approach of using mixed criticality scheduling in CPSs has been mainly application related, leading to a variety of particular task models with a multitude of system behavior constraints (Burns and Davis, 2017). Such methods, even while featuring a high practical relevance, have shortcomings when it comes to scalability and interaction with an unpredictable environment.

This paper proposes a systematization of the mixed criticality task scheduling algorithms, trying to narrow the gap between theory and practice and between the approaches from different fields, in order to ease the integration of more general mixed criticality system concepts and techniques into cyber physical systems.

The main contributions of this paper are:

- An overview of the state-of-the-art mixed criticality scheduling algorithms, both for single and multiple processing units.
- A comparison of the most notable scheduling algorithms, with a focus on the ones suitable for CPSs.
- Emphasizing the necessity to incorporate MCSs in CPSs.
- A discussion about the advantages and disadvantages of using mixed criticality scheduling algorithms in CPSs.
- Identifying the challenges and future perspectives of mixed criticality scheduling in CPSs.

The remainder of this paper is structured as follows: Section 2 describes the particularities of CPSs and their influence on scheduling mechanisms, while Section 3 presents the standard MCS task model. Then, Section 4 reviews and compares the most notable scheduling algorithms in the field. In Section 5, the research gaps for applying MC scheduling algorithms to CPSs are highlighted and future perspectives are discussed. The paper ends with conclusions in Section 6 and some proposed approaches for CPS areas of research.

2. Constructive and functional features of CPSs as processing units

Our work aims to facilitate the integration of MC scheduling approaches in CPSs by providing an analysis of the state-of-the-art mixed criticality scheduling algorithms. Even though CPSs include a wide range of systems, very different from the architectural and functional perspective, one can identify common attributes (Rodriguez et al., 2013). Each of these features, further listed in this section, have a certain influence on the scheduling mechanism.

2.1. Heterogeneity

In the context of CPSs, heterogeneity refers to different hardware specifications, different application and power consumption requirements. All these aspects can be modeled by various parameters, as reflected in the mixed criticality scheduling theory.

2.2. Power management

Because numerous components in CPSs are mobile, special care must be provided to the power management aspect, resulting in the development of many specific power aware scheduling algorithms, both for classical real-time systems and for MCSs (Rodriguez et al., 2013; Fakih et al., 2017; Taherin et al., 2018).

2.3. Dynamism and self-adaptability

Due to the direct interaction with the physical environment, which is often dynamic and unpredictable, the system must be able to face changes in a real-time manner (Gerostathopoulos et al., 2016; Micea et al., 2017). Using only static scheduling mechanisms can be insufficient in this case.

2.4. Robustness

Since the environment in which CPSs operate is usually subject to several uncertainties in terms of run-time behavior (network unavailability, hardware failures, etc.), it is also desirable that critical applications are not affected by the failures or computation overload caused by any other application. This aspect is reflected in the need of adding hypervisor mechanisms or monitors.

2.5. Redundancy

In order to avoid single point of failure, both hardware and computational redundancies are desirable and, therefore, dynamic scheduling policies are needed, to adapt to different computation loads and reschedule newly arrived applications.

2.6. Distribution

CPSs usually have different components which communicate and interact with each other (for example in the case of robotic or wireless sensor network applications). Depending on the location of the components and on the communication environment between them, one must consider scheduling algorithms for multiple processing units. Hierarchical design, with one or more arbiters that will coordinate different subsystems, might also be required in some cases.

2.7. Scalability

When considering systems with multiple processing units, their scalability in terms of hardware components, application design, analysis, coding and testing is of relatively great concern. Therefore, scalability of the task models, scheduling mechanisms and of the scheduling analysis must also be provided.

2.8. Security

As all computational systems, CPSs can be also subject to malicious attacks. Moreover, a complex system with various functionalities of different criticality levels will be harder to protect. Increased caution must be given especially regarding the critical applications, thus, a direct implication on the scheduling policy is that the critical applications must be isolated from the less critical ones, from which arises the isolation concern, essential in MCSs.

2.9. Isolation

In a complex system, containing different functionalities, critical components must be isolated from the non-critical ones in terms of temporal behavior and resource usage.

Designing scheduling mechanisms which have these attributes is a challenging problem. Even though a common technique that can solve these issues cannot be found, mainly because of the vast applicability of CPSs in different fields, this paper aims to assist in choosing the best scheduling algorithm to minimize these problems and achieve high performance and schedulability.

3. MCS task model

One noticeable difference between CPSs applied in different areas is the way the software components or the functionalities are modeled. In some areas, there are processes running in continuous

fashion, while in others, multiple functionalities run as different software entities on the same hardware or even software as a service (Lee and Shin, 2017).

In mixed criticality systems, as in real-time systems, the basic level of an application is represented by the task. A task is an abstract concept representing the basic unit of software execution. One of the most important aspects modeled by the tasks in real-time and MCSs is the temporal behavior. While in real-time and MCSs there are different task models, still some of the parameters are common (Schneider et al., 2013):

- Period T_i - the (minimum) arrival interval between two consecutive jobs of the same task i.
- Deadline D_i - the time by which any job execution needs to complete, relatively to its arrival time.
- Computation time C_i - the worst case execution time for any job of task i.
- Release time r_i – the earliest time when a job can start its execution, expressed relatively to the start time of the system.

The temporal behavior of a task is always modeled by discreet values, even if the controlled process is a physical one, having a continuous behavior. As stated in Stigge and Yi (2015), performing control loop operations in a continuous fashion is not feasible, thus all the operations are performed at discrete time intervals. How to translate the characteristics from continuous functions into discrete time parameters is explained extensively in Stigge and Yi (2015).

If the application components have precedence requirements, beside the task temporal parameters there is a precedence graph associated with the task set. A survey of the most used models of such task sets is available in Stigge and Yi (2015). Thus, these types of models are more complex and reach another level of the application.

At a higher level of the application there can be sets of tasks or meta-tasks (Braun et al., 1999), with or without precedence or temporal requirements at the task set level.

On top of the task model, presented above, Vestal proposes in Vestal (2007) a task model for MCSs, basically extending the general task model, by adding a new parameter reflecting the criticality level of each task. The extension leads to the following set of parameters:

- Criticality level L_i
- Period (minimum arrival interval) T_i
- Deadline D_i
- Computation time C_i (vector of values – one per criticality level, for levels lower or equal to the criticality level L_i, expressing the worst case execution time for each criticality level).

Based on this task model and on other extensions of it, numerous task scheduling algorithms have been developed. Some of the most relevant algorithms are synthesized in the following section.

4. MCS scheduling

In cyber-physical systems, especially in real-time systems, task scheduling is the key mechanism to ensure the imposed performance of the system regarding temporal behavior. In the case of the mixed criticality applications, things get more complicated as the number of variables which need to be taken into consideration increases (Asyaban and Kargahi, 2017).

As in the case of the real-time systems, the scheduling algorithms can be classified into single processing unit and multiple processing units, depending on the targeted platform. On the other hand, the scheduling algorithms can be classified into static algorithms, if the scheduling decisions are done offline (before the system starts running), dynamic algorithms, if the scheduling decisions are made during runtime at different time moments, or hybrid, if the scheduling decision is partially taken offline and partially online.

Next, we will discuss some of the scheduling algorithms developed for single and multiple processing units in mixed criticality systems.

4.1. Processing unit level scheduling algorithms

4.1.1. Classification

Regarding the algorithms running on a single processing unit or at the unit level in a system with multiple processing units, they can be classified according to the way the priority is assigned to the task instances (jobs) (Crespo et al., 2014).

- Fixed Task-Priority (FTP) class - All the jobs generated by a given task are assigned the same priority.
- Fixed Job-Priority (FJP) class - Different jobs of the same task may have different priorities. However, the priority of each job may not change between its arrival time and its completion time.
- Dynamic Priority (DP) class - Priorities of jobs may change between their release times and their completion times.
- Hybrid Priority (HP) class – The scheduling policies incorporate features of multiple scheduling algorithm classes.

FJP scheduling represents a generalization of FTP scheduling, and DP scheduling represents a generalization of FJP scheduling.

These scheduling algorithms have different approaches for dealing with the following challenges:

4.1.2. Task/job priority assignment

Assigning priorities to different task instances must take into consideration, beside the temporal parameters such as deadline or period, also the criticality level of the task. In order to achieve this, several solutions have been proposed (Baruah et al., 2011a):

- To use the criticality levels for priority assignment: criticality monotonic priority assignment or Own Criticality Based Priority (OCBP) (Crespo et al., 2014; Baruah et al., 2011a). With this approach all jobs of high criticality have a higher priority than all jobs of any lower criticality level. Within a criticality level, priorities are assigned according to a standard optimal scheme such as the deadline rate monotonic priority assignment.
- To make use of a technique, whose concept originated in the classical real-time systems, called Period Transformation (PT). This technique basically transforms the original period of the task, to reflect both the temporal behavior and the criticality of that task. In this way priorities of different criticality jobs are interleaved, e.g. EDF-VD (Arlock and Linderoth-Olson, 2014) and EDF-DB (Baruah et al., 2011a).
- To make use of Static/Adaptive Mixed Criticality (SMC/AMC) algorithms based on Audsley's priority assignment algorithm (Crespo et al., 2014).
- To make use of heuristic algorithms for priority assignment (Asyaban and Kargahi, 2017).

4.1.3. Schedulability tests

Regarding the schedulability tests, most of the algorithms, for both static and adaptive MCSs, were tested by applying response time analysis (RTA) (Burns and Davis, 2017), especially in the case of static priority assignment and, based on demand bound function (DBF) (Rodriguez et al., 2013), in the case of dynamic priority assignment. An exact schedulability test is also available in Asyaban and Kargahi (2017) for the fixed priority preemptive scheduling algorithms running on a single processing unit.

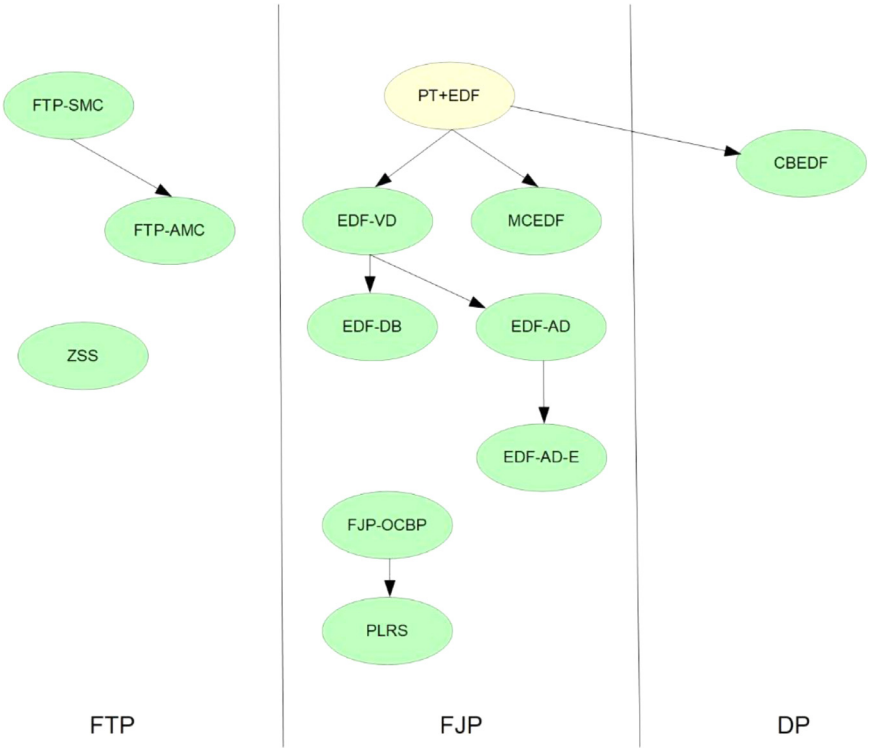

Fig. 1. A classification of the scheduling algorithms for single processing units.

The schedulability tests mentioned in this article analyze the schedulability in different criticality modes, but not during the criticality mode change.

Fig. 1 illustrates a graphical representation of the algorithms classified in Table 1, following the one proposed in Müller and Werner (2011) for classical real-time systems. In the figure, the arrow pointing towards an algorithm specifies that it is based on an already existing scheduling technique. Yellow symbolizes a classical scheduling algorithm, while green refers to mixed criticality approaches.

Because of the lack of benchmarks and common metrics suitable for all the algorithms presented so far, an exact comparison between them is infeasible. Considering the CPSs attributes outlined in Section 2, next, we define a series of compliance levels.

Table 3 presents the evaluation for single processing unit scheduling algorithms based on the attribute levels introduced in Table 2.

4.2. Multiple processing units scheduling algorithms

4.2.1. Class P: partitioned schedulers

In partitioned scheduling each task is assigned to a single processing unit. Some advantages of this scheduling method are (Crespo et al., 2014):

- Deadline violations can only affect tasks that are on the same processing unit.
- There is no migration penalty.
- Each processing unit uses a separate run queue which helps reduce overhead.

Partitioned scheduling has a few disadvantages as well (Crespo et al., 2014):

- Some task sets are only schedulable if migration (between processing units) is present.
- Every task set needs a new optimal allocation to the processing unit.
- Processing units are often idle. They cannot be used by tasks that are allocated to a different processing unit.

A large area of research has addressed partitioned schedulers for mixed criticality systems (Capota et al., 2018). Baruah et al. (2014) described and evaluated a set of algorithms for partitioned scheduling on multiple processing units, namely MC-PARTITION. Some pragmatic improvements have also been developed: MC-PARTITION-UT-0.75, MC-PARTITION-UT-1 and MC-PARTITION-UT-INC. The latter version dominates the first two. This algorithm and its improvements use EDF-VD as processing unit level scheduler.

Two real-time partitioned schedulers are also presented in Lu (2016), namely the Time Triggered Scheduler with Mode Change (TTS-MC) and the Event Scheduler in Multi-Core. For the TTS-MC, each core has its own Time Triggered Scheduler, which is only in charge of scheduling the tasks assigned to it. Similarly, for the Event Scheduler-MC, every core has its own Event Scheduler.

Another partitioned scheduler is described in Giannopoulou (2017), Huang et al. (2015) and Giannopoulou et al. (2017). MC-IS-Server extends the IS-Server strategy presented in the same papers, to mixed criticality systems. The algorithm can be applied to systems with two or more criticality levels. A joint task partitioning and deadline shortening heuristic called Mixed-Critical EY Worst Fit (MC-EY-WF) is used. Two global servers are considered: one for "HI" criticality tasks and one for "LO" criticality tasks. When the system is in LO mode, tasks are scheduled within each server in a manner corresponding to partitioned EDF, using the shortened deadlines for HI criticality tasks. When a switch to HI mode occurs, the LO server is disabled, while the remaining HI criticality tasks are scheduled according to partitioned EDF, using their original deadlines.

Table 1
Scheduling algorithms for single processing units.

Scheduling Algorithm	Class	Priority assign-ment process	Priority assignment policy	Reference implemen-tation	Schedulability Test	Schedulability Test Type	Ref.
Fixed Task-Priority Static Mixed-Criticality (FTP-SMC)	FTP	static	Audsley	Simulator/Linux	RTA based: SMC-NO	sufficient	(Burns and Davis, 2017), (Arlock and Linderoth-Olson, 2014), (Baruah et al., 2011b)
Fixed Task-Priority Adaptive Mixed-Criticality (FTP-AMC)	FTP	static	Audsley	Simulator/Linux	RTA based: AMC-RT AMC-MAX	sufficient	(Burns and Davis, 2017), (Arlock and Linderoth-Olson, 2014), (Baruah et al., 2011b)
Fixed Task-Priority preemptive (FTP-preemptive)	FTP	static	heuristic	Simulator	SB-RTA	exact	(Asyaban and Kargahi, 2017)
Zero-Slack Scheduling (ZSS)	FTP	static	RM	Simulator/Linux	RTA/Slack based	sufficient	(Arlock and Linderoth-Olson, 2014), (Lakshmanan et al., 2011)
Fixed Job-Priority Own Criticality Based Priority (FJP-OCBP)	FJP	static	Audsley	Theoretical	DBF based	sufficient	(Arlock and Linderoth-Olson, 2014), (Santy et al., 2012), (Baruah and Guo, 2015), (Völp et al., 2014)
Earliest Deadline First-Virtual Deadlines (EDF-VD)	FJP	dynamic	EDF	Simulator/Linux	DBF based	sufficient	(Burns and Davis, 2017), (Arlock and Linderoth-Olson, 2014), (Baruah et al., 2012), (Burns, 2015), (Ali et al., 2015), (Huang et al., October)
Earliest Deadline First-Demand Bound (EDF-DB)	FJP	dynamic	EDF	Simulator	DBF based	sufficient	(Arlock and Linderoth-Olson, 2014), (Ekberg and Yi, 2014)
Earliest Deadline First-Adaptive Task Dropping (EDF-AD)	FJP	dynamic	EDF	Simulator	DBF based	sufficient	(Lee et al., 2017)
Earliest Deadline First-Adaptive Task Dropping-Enhanced (EDF-AD-E)	FJP	dynamic	EDF	Simulator	DBF based	sufficient	(Lee et al., 2017)
Criticality Based Earliest Deadline First (CBEDF)	DP	dynamic	EDF	Simulator	DBF based	sufficient	(Park and Kim, 2011, October)
Priority List Reuse Scheduling (PLRS)	FJP	hybrid	OCBP	Simulator	DBF based	sufficient	(Arlock and Linderoth-Olson, 2014), (Guan et al., 2011)

Table 2
Levels of compliance.

Attribute	Levels	Level description
Heterogeneity	At task level	Handles different task set types
	At device level	Handles different architectures
Power management	–	Does not consider power management
	Low	Considers static power consumption
	Moderate	Considers dynamic power consumption
	High	Considers static and dynamic power consumption
Dynamism and self-adaptability	Low	Does not accept dynamic task loading
	Moderate	Accepts limited dynamic task loading
	High	Task sets can be loaded dynamically during run time
Robustness	Low	Does not handle overload
	Moderate	Handles overload only for high criticality levels
	High	Handles overload for both low and high criticality levels
Distribution	Yes/No	Algorithms for distributed systems/Other algorithms
Scalability	Yes/No	Scalable with respect to the number of criticality levels/Not scalable
Security and Isolation	Low	Only temporal isolation between criticality levels
	Moderate	Temporal and spatial isolation only for high criticality levels
	High	Temporal and spatial isolation between different criticality levels

4.2.2. Class G: global schedulers

Under this class, tasks can migrate from one processing unit to another. The main advantages of global scheduling are (Crespo et al., 2014):

- Fewer context switches, because a preemption will occur only when there are no idle processing units.
- There is no need for allocation algorithms when the set of tasks changes.
- The processing units are better utilized as tasks can migrate from one processing unit to another.

The most significant problems are (Crespo et al., 2014):

Table 3
Attribute levels of scheduling algorithms in single processing units.

Scheduling Algorithm	Power management	Dynamism and self-adaptability	Robustness	Distribution	Scalability	Security
Fixed Task-Priority Static Mixed-Criticality (FTP-SMC)	–	Low	Low	No	Yes	Low
Fixed Task-Priority Adaptive Mixed-Criticality (FTP-AMC)	–	Low	Moderate	No	Yes	Low
Fixed Task-Priority preemptive (FTP-preemptive)	–	Low	Moderate	No	Yes	Low
Zero-Slack Scheduling (ZSS)	–	Low	Moderate	No	Yes	Low
Fixed Job-Priority Own Criticality Based Priority (FJP-OCBP)	High	Low	Low	No	Yes	Low
Earliest Deadline First-Virtual Deadlines (EDF-VD)	High	High	Moderate	No	Yes	Low
Earliest Deadline First-Demand Bound (EDF-DB)	–	High	Moderate	No	Yes	Low
Earliest Deadline First-Adaptive Task Dropping (EDF-AD)	–	High	Moderate	No	Unspecified	Low
Earliest Deadline First-Adaptive Task Dropping-Enhanced (EDF-AD-E)	–	High	Moderate	No	Unspecified	Low
Criticality Based Earliest Deadline First (CBEDF)	–	High	High	No	No	Low
Priority List Reuse Scheduling (PLRS)	–	Moderate	Low	No	Yes	Low

- Migration of tasks to another processing unit is permitted. This causes a high overhead in the system.
- It might involve the use of shared memory and communication channels because global scheduling increases the communication flow between processing units.
- It uses a single queue for all processing units.
- Low predictability.
- Poor performance for some particular task sets.

The research published by Baruah et al. (2014) introduces an algorithm which extends the approach for single processing units presented in Baruah and Guo (2015) to multiple processing units, by applying the global scheduling algorithm fpEDF (Baruah, 2004) (for non MC) to mixed criticality systems. An improved version of the algorithm has also been described, GLOBAL-PRAGMATIC in the same paper. The tasks assigned to each processing unit are scheduled by EDF-VD in both cases.

Lee et al. (2014) proposed a scheduling technique in multiple processing units for dual-criticality task systems called MC-Fluid. Similar to EDF-VD, MC-Fluid takes into consideration only two levels of criticality (HI and LO), where the deadlines of its HI criticality tasks are shortened in LO mode. In both LO and HI modes, tasks are scheduled by another technique, DP-Fair (Funk et al., 2011). MC-IS-Fluid (Huang et al., 2015; Giannopoulou et al., 2017) extends the MC-Fluid algorithm to include isolation between criticality levels. It also provides support for more than two criticality levels.

4.2.3. Class C: clustered/semi-partitioned schedulers

A clustered scheduler is a hybrid approach between the partitioned and global schedulers which refers to a group of processing units where each cluster is divided into sub-clusters. This method has the following benefits (Awan et al., 2017; Ali and Kim, 2017):

- Basic tasks are grouped into subsets that are assigned to processing units and executed sequentially, resulting in zero intra-cluster overhead.
- Reduces the parallel execution time.
- Different global scheduling algorithms can be used to schedule the tasks in a cluster.
- Reduces migration costs, as most tasks are partitioned under semi-partitioned scheduling and the rest may migrate in a well managed manner.
- Improves processor utilization compared to the partitioned approach.
- Favors large scale platforms with multiple processing units.

On the other hand, it presents some difficulties (Awan et al., 2017; Ali and Kim, 2017):

- Small cluster sizes can suffer from bin-packing limitations in high criticality modes.
- Relatively high computational complexity.

Ali and Kim (2017) were the first to propose a cluster-based scheduling scheme for real-time mixed criticality systems with multiple processing units. The approach uses smaller cluster sizes (sub-cluster) in low criticality mode because the utilization of each mixed criticality task is smaller, and larger cluster sizes in high criticality mode, due to the increase in the utilization of each high-criticality task. Low-criticality tasks stop executing when a mode switch takes place from low to high criticality. All the low and high criticality tasks in sub-clusters are initially scheduled in low criticality mode using a global fixed-priority algorithm. Furthermore, the mixed criticality task set is arranged in decreasing criticality - decreasing utilization (DCDU) order, while allocating tasks to clusters is done by using the Worst Fit heuristic (Burke et al., 2006).

4.2.4. Class D: distributed schedulers

Scalability is an important feature to consider in CPSs, which consequently leads towards a distributed approach, due to the fact that centralized scheduling is not feasible for coordinating multiple components in a dynamic environment.

Because the environment is dynamic and the system can be composed of multiple processing units, distributed schedulers have the highest potential in CPSs (Zhao et al., 2018). They encapsulate scheduling techniques for both single and multiple processing units, thus distributed algorithms can be seen as complex but more practical for integrating physical and computational capabilities on the same platform.

Distributed schedulers (Zhan et al., 2018) are developed for systems in which the components are spread out over multiple processing units, using mixed criticality communication networks. The main feature of a distributed scheduler is that it uses distributed middleware in order to interconnect partitions. A partition can have one or multiple processing units.

Main disadvantages regarding the distributed scheduler class are:

- Increased complexity
- Additional resource allocation problems
- Requires spatial and temporal partitioning

The main advantages of distributed schedulers are:

- High performance, flexibility, adaptability and energy efficiency

Moreover, CPSs encapsulate multiple components with different specifications, application system requirements and hardware

configurations, therefore distributed schedulers on heterogeneous systems render a substantial performance improvement compared to a centralized approach. This type of scheduling is optimal for CPSs, even though it presents limitations on resource sharing. There are various forms of heterogeneity: configurational, which involves different application and power consumption requirements; architectural, concerning system capabilities and lastly, operating system heterogeneity, as different processing units have different operating system configurations (Zhou et al., 1993).

A notable example of research for Cyber Physical Systems regarding this subject was made by Pérez et al. (2017). The paper describes a partitioned distributed real-time platform that incorporates hypervisor techniques and standard distributed middleware. Furthermore, the proposed distributed architecture for mixed criticality multi-core platforms uses the XtratuM (Crespo et al., 2010) hypervisor and the DDS (Data Distribution Service) standard (Object Management Group 2007). The latter is a middleware that relies on a publisher-subscriber communication pattern, where data, defined by topics, can flow between publisher and subscriber entities within a global data space. Subscribers must specify their interest to receive a particular topic, and communication is allowed only between publishers and subscribers from the same domain. The XtratuM (Crespo et al., 2010) hypervisor uses para-virtualization techniques to provide virtual CPUs to the partitions and can assign the same scheduling policy to one or more processing units.

Some cross-domain patterns for mixed criticality systems have been presented in Larrucea et al. (2016). The patterns are developed under the European project DREAMS. Its architecture is based on nodes which are composed of application tiles. Furthermore, each tile contains one or more partitions with different criticality levels. The cross-domain patterns discussed, describe reusable generic solutions for hypervisors, Commercial-Off-The-Shelf (COTS) multi-core devices and mixed criticality networks.

A criticality-aware bin-packing algorithm, called Compress-on-Overload Packing (COP), was introduced in Lakshmanan et al. (2010), as an extension to the zero-slack rate-monotonic (ZSRM) scheduler (De Niz et al., 2009). This algorithm aims at maximizing the ductility metric in distributed mixed criticality cyber physical systems. Ductility characterizes the behavior of a system under overload from the perspective of resource allocation. The metric ensures that high criticality tasks meet their deadlines even under system overload.

In a system where both critical and non-critical applications coexist, it is important to enforce spatial and temporal separation. A simulated annealing-based technique (De Niz et al., 2009) was proposed for optimizing time slots in mixed-critical real-time heterogeneous embedded systems. The approach uses a partitioning scheme, so that each application runs on a different partition, with an allocated number of time slots. The algorithm was applied on a system where safety-critical applications are scheduled using static-cycling scheduling and the non-critical applications use fixed-priority preemptive scheduling.

Xie et al. (2016) proposed another scheduling algorithm for mixed-critical heterogeneous distributed embedded systems, namely the deadline-span of multiple heterogeneous earliest finish time (D_MHEFT). The goal in this case is for more high-criticality functions to finish execution before their deadlines, resulting in a low deadline miss ratio. The system's criticality is modified in order to schedule correctly functions that have higher or equal criticality levels compared to the system.

Two efficient scheduling algorithms for automotive cyber-physical systems (ACPS) are introduced by Xie et al. (2017), namely a fairness-based dynamic scheduling algorithm FDS_MIMF and an adaptive dynamic scheduling algorithm ADS_MIMF. FDS_MIMF was developed to minimize individual makespans of functions from a high performance perspective in order to meet the requirements of heterogeneity, dynamics and parallelism in ACPS. ADS_MIMF brings some improvements by responding to additional challenges such as safety and criticality. The algorithm allows achieving low deadline miss ratio (DMR) values of high-criticality functions while keeping an acceptable performance of ACPS.

Distributed scheduling algorithms can be partitioned or global based (D_MHEFT, F_MHEFT). There are two types of scheduling: static (D_MHEST, F_MHEFT) and dynamic (FDS_MIMF, ADS_MIMF). In static scheduling functions are released simultaneously while in dynamic scheduling they are released at different time instances.

Fig. 2 and Table 4 illustrate the above specified scheduling techniques for multiple processing units. In the figure, the arrow pointing towards an algorithm specifies that it is based on an already existing scheduling technique. Orange symbolizes a classical scheduling algorithm, while green refers to mixed criticality approaches.

Our next milestone is the analysis of how the main features of CPSs (already presented in Section 2) can be covered by notable mixed criticality scheduling algorithms. Thus, Table 5 comparatively illustrates the adequacy of notable MCS algorithms in the context of CPSs, using the levels defined in Table 2 for the scheduling algorithms in multiple processing units described previously. The idea behind this analysis is to either help incorporate some of the already existing MC scheduling methods in CPS areas or to use these scheduling schemes as a basis for developing other, more feasible algorithms.

As previously mentioned, distributed scheduling algorithms are the most promising for CPSs due to their scalability, dynamism and self-adaptability, which are key aspects in an ever changing environment. F_MHEFT and D_MHEFT (Xie et al., 2016) were initially developed for heterogeneous distributed embedded platforms but in Xie et al. (2016) they are discussed from the perspective of automotive systems. Nevertheless, both scheduling methods could be integrated into other CPS areas as well, such as industrial systems or internet of things. The FDS_MIMF and ADS_MIMF algorithms (Xie et al., 2017) were mainly developed for automotive cyber physical systems, in order to meet certain specific demands in the field. FDS_MIMF can also be applied to avionics and industrial systems, while ADS_MIMF is suitable for domains with very strict temporal constraints and security requirements (avionics and medical devices).

5. Integrating MCSs in CPSs: research challenges, advantages and future perspectives

The idea of integrating MCSs in CPSs has been considered before, however not on a large scale. Several scheduling algorithms were proposed, but there have not been any standardized models thus far. As mentioned by Lee (2008) temporal behavior is the main issue when designing CPSs. This becomes harder to predict as the complexity of the system increases and communication takes place between real-time and non-real-time components.

The CPS concept is still in its infant stages, as it faces some barriers due to a large diversity in requirements and challenges for each area of development.

Fig. 3 highlights the growing interest for MC techniques in CPS domains. Certain fields such as automotive and avionics have more generic scheduling algorithms whereas for medical devices they are more application specific, thus integrating MC approaches in such fields can prove to be challenging, requiring further research. The information extracted from the Web of Science (www.webofknowledge.com/) (searching the exact phrase "mixed criticality" together with each of the domain names presented in the legend of Fig. 3) covers two time spans. A column represents the percentage of papers which discuss mixed criticality techniques in

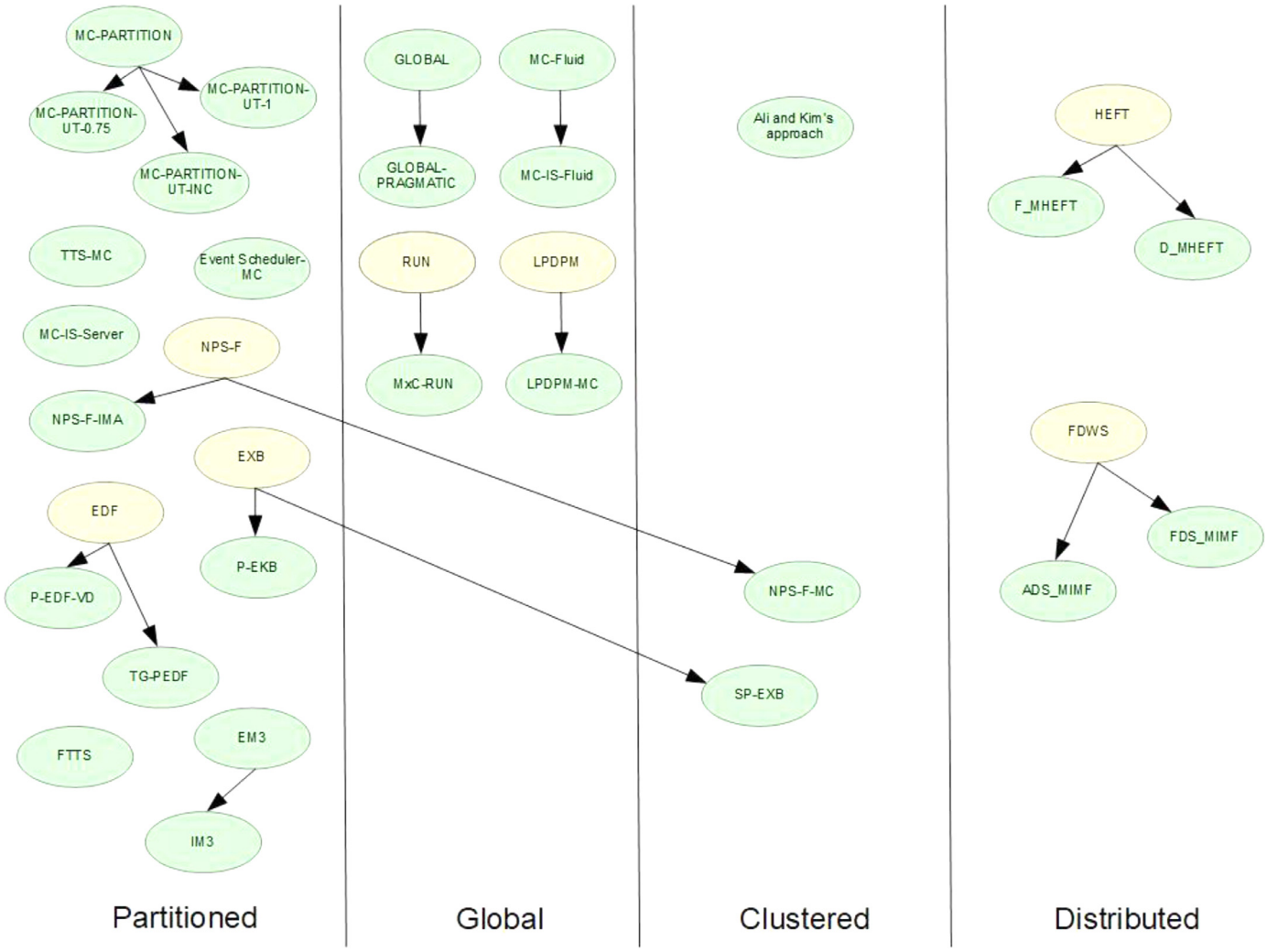

Fig. 2. A classification of relevant scheduling algorithms for multiple processing units.

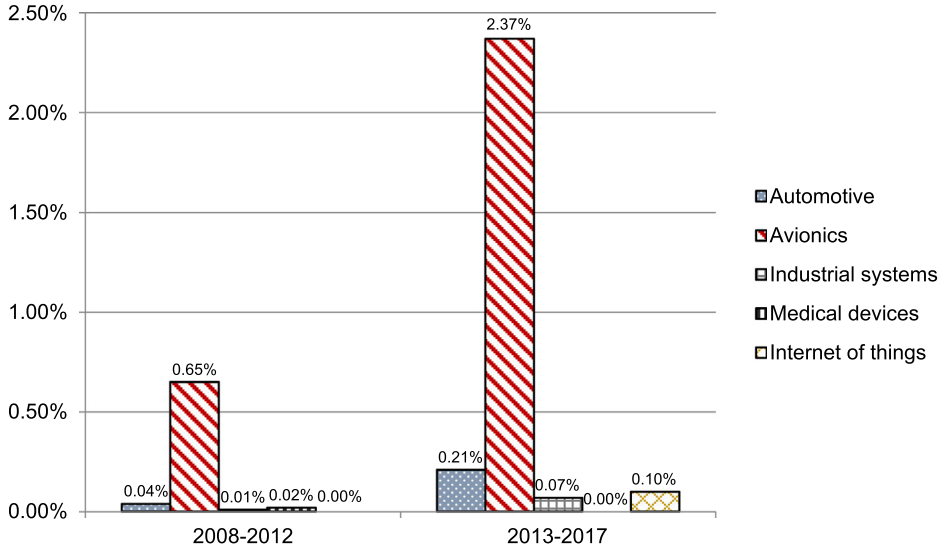

Fig. 3. The current interest in the research of mixed criticality concepts for cyber physical systems, with data extracted from the Web of Science.

Table 4
Scheduling algorithms for multiple processing units.

Scheduling Algorithm	Class	Reference implementation	Task migration	Ref.
Mixed-Criticality-PARTITION (MC-PARTITION)	P	Simulator	No	(Baruah et al., 2014)
Mixed-Criticality PARTITION-Utilization-0.75 (MC-PARTITION-UT-0.75)	P	Simulator	No	(Baruah et al., 2014)
Mixed-Criticality PARTITION-Utilization-1 (MC-PARTITION-UT-1)	P	Simulator	No	(Baruah et al., 2014)
Mixed-Criticality PARTITION-INCREMENT (MC-PARTITION-UT-INC)	P	Simulator	No	(Baruah et al., 2014)
Time-Triggered Scheduler with Mode Change (TTS-MC)	P	Simulator/Linux	No	(Lu, 2016)
Event Scheduler in Multi-Core	P	Simulator/Linux	No	(Lu, 2016)
Mixed-Criticality Isolation Server (MC-IS-Server)	P	Simulator	No	(Giannopoulou et al., 2017)
Task Grouping-Partitioned Earliest Deadline First (TG-PEDF)	P	Simulator	No	(Ren and Phan, 2015)
Notional Processor Scheduling-Fractional Capacity-Integrated Modular Avionics (NPS-F-IMA)	P	Simulator	No	(Awan et al., 2017)
Partitioned-Eckberg (P-EKB)	P	Simulator	No	(Awan et al., 2017)
Partitioned Earliest Deadline First-Virtual Deadlines (P-EDF-VD)	P	Simulator/Linux	No	(Han et al., 2018)
Flexible Time-Triggered Scheduling (FTTS)	P	Simulator/Kalray MPPA-256 Andey many-core platform	No	(Ren and Phan, 2015)
Energy Minimized Mixed-Criticality (EM3)	P	Simulator	No	(Narayana et al., 2016)
Isolated Mixed-Criticality (IM3)	P	Simulator	No	(Narayana et al., 2016)
Ali and Kim's approach	C	Simulator	Yes	(Ali and Kim, 2017)
Notional Processor Scheduling-Fractional Capacity-Mixed-Criticality (NPS-F-MC)	C	Simulator	Yes	(Awan et al., 2017)
Semi-Partitioned-Eckberg (SP-EKB)	C	Simulator	Yes	(Awan et al., 2017)
Mixed-Criticality-Reduction to Uniprocessor (MxC-RUN)	G	Simulator	Yes	(Gratia et al., 2015)
GLOBAL	G	Simulator	Yes	(Baruah et al., 2014)
GLOBAL-PRAGMATIC	G	Simulator	Yes	(Baruah et al., 2014)
Mixed-Criticality-Fluid (MC-Fluid)	G	Simulator	Yes	(Lee et al., 2014)
Mixed-Criticality Isolation-Fluid (MC-IS-Fluid)	G	Simulator	Yes	(Giannopoulou et al., 2017)
Linear Programming Dynamic Power Management-Mixed-Criticality (LPDPM-MC)	G	Simulator	Yes	(Legout et al., 2013)
Fairness on Multiple Heterogeneous Earliest Finish Time (F_MHEFT)	D	Simulator	Yes	(Xie et al., 2016)
Deadline-span of Multiple Heterogeneous Earliest Finish Time (D_MHEFT)	D	Simulator	Yes	(Xie et al., 2016)
Fairness-based Dynamic Scheduling-Minimize Individual Makespans of Functions (FDS_MIMF)	D	Simulator	Yes	(Xie et al., 2017)
Adaptive Dynamic Scheduling-Minimize Individual Makespans of Functions (ADS_MIMF)	D	Simulator	Yes	(Xie et al., 2017)

each CPS area from the total amount of articles published during a certain time span. Even though an increase in the number of published articles is visible throughout the two time spans, it has a slow ascension due to numerous challenges which we will discuss next.

5.1. Challenges and open issues in mixed criticality systems

Because MCSs are in a continuous development and, currently, there is insufficient research in the field, a set of important challenges exist:

- **Isolation** – this is the most important aspect in MCSs. Thus far, research has focused mainly on application/system specific solutions. Contributions have been made in an attempt to solve these challenges on different domains. Beside the application/domain specific solutions discussed in Section 4, a more generic example is the LITMUS (Calandrino et al., 2006) real-time extension for the Linux kernel which is used to evaluate different scheduling algorithms from a research point of view. An important requirement in MCS scheduling is that tasks of lower criticality should not interfere with tasks of higher criticality, which is known as *temporal isolation*. LITMUS ensures temporal isolation between components, so that the timing requirements can be validated independently. Isolated solutions like LITMUS for Linux based systems can be further extended to include other embedded or real-time operating systems.
- **Unitary models** – another concern appears from the lack of unitary models in CPS domains such as task level and scheduling level models. Some domains do not have a standardized approach for scheduling, such as medical (Cavalcanti and Freitas, 2005) or Internet of Things (Accettura et al., 2013). This gives rise to problems when integrating MCSs in CPSs, because a unitary approach requires well defined general scheduling algorithms, which can be easily customized according to the application environment.
- **Task scheduling, in the context of criticality level isolation and efficiency** – as already stated, the core concept for developing a MCS is the demonstration of sufficient independence between the criticality levels. Task scheduling in MCSs is a challenging issue due to several factors like: priority assignment, considering both the criticality level and the time specifications, criticality level isolation (an execution flow of a low criticality task must not affect higher criticality tasks execution), contention of shared resources and resource utilization efficiency.
- **Real-time communication** – in a complex system with real-time and non-real-time components, real-time communication must be ensured between all the components. This means that, in order to have a real-time distributed system, all the components must respect the real-time paradigm. Therefore, having a real-time communication support is imperative.
- **Resource sharing** – this is harder to be achieved in the context of time and functionality isolation. High criticality tasks gain precedence in resource allocation over low criticality tasks.

Other challenges, which come from the specificity of the cyber-physical systems, may hinder the process of integrating MCSs in CPSs:

Table 5
Attribute levels of different scheduling algorithms in multiple processing units.

Scheduling Algorithm	Hetero-geneity	Power management	Dynamism and self-adaptability	Robustness	Distribution	Scalability	Security
Mixed-Criticality-PARTITION (MC-PARTITION)	At task level	–	Moderate	Moderate	No	Unspecified	Low
Mixed-Criticality PARTITION-Utilization-0.75 (MC-PARTITION-UT-0.75)	At task level	–	Moderate	Moderate	No	Unspecified	Low
Mixed-Criticality PARTITION-Utilization-1 (MC-PARTITION-UT-1)	At task level	–	Moderate	Moderate	No	Unspecified	Low
Mixed-Criticality PARTITION-INCREMENT (MC-PARTITION-UT-INC)	At task level	–	Moderate	Moderate	No	Unspecified	Low
Time-Triggered Scheduler with Mode Change (TTS-MC)	At task level	–	Moderate	Moderate	No	Unspecified	Low
Event Scheduler in Multi-Core	At task level	–	Moderate	Moderate	No	Unspecified	Low
Mixed-Criticality Isolation Server (MC-IS-Server)	At task level	–	Moderate	Moderate	No	Yes	Low
Task Grouping-Partitioned Earliest Deadline First (TG-PEDF)	At task level	–	Moderate	High	No	Yes	Moderate
Notional Processor Scheduling-Fractional Capacity-Integrated Modular Avionics (NPS-F-IMA)	At task level	–	Moderate	High	No	Unspecified	High
Partitioned-Eckberg (P-EKB)	At task level	–	Moderate	Moderate	No	Unspecified	Low
Partitioned Earliest Deadline First-Virtual Deadlines (P-EDF-VD)	At task level	–	Moderate	Moderate	No	Yes	Low
Flexible Time-Triggered Scheduling (FTTS)	At task level	–	Low	Moderate	No	Yes	Low
Energy Minimized Mixed-Criticality (EM3)	At task level	High	Moderate	Moderate	No	Unspecified	Low
Isolated Mixed-Criticality (IM3)	At task level	High	Moderate	High	No	Unspecified	High
Ali and Kim's approach	At task level	–	Moderate	Moderate	No	Yes	Low
Notional Processor Scheduling-Fractional Capacity-Mixed-Criticality (NPS-F-MC)	At task level	–	High	High	No	Unspecified	High
Semi-Partitioned-Eckberg (SP-EKB)	At task level	–	High	Moderate	No	Unspecified	Low
Mixed-Criticality-Reduction to Uniprocessor (MxC-RUN)	At task level	–	High	Moderate	No	Unspecified	Low
Linear Programming Dynamic Power Management-Mixed-Criticality (LPDPM-MC)	At task level	Low	Low	High	No	Unspecified	Low
GLOBAL	At task level	–	High	Moderate	No	Unspecified	Low
GLOBAL-PRAGMATIC	At task level	–	High	Moderate	No	Unspecified	Low
Mixed-Criticality-Fluid (MC-Fluid)	At task level	–	High	Moderate	No	Unspecified	Low
Mixed-Criticality Isolation-Fluid (MC-IS-Fluid)	At task level	–	High	Moderate	No	Yes	Low
Fairness on Multiple Heterogeneous Earliest Finish Time (F_MHEFT)	At the device level	–	Low	Low	Yes	Yes	Unspecified
Deadline-span of Multiple Heterogeneous Earliest Finish Time (D_MHEFT)	At device level	–	Low	Moderate	Yes	Yes	Unspecified
Fairness-based Dynamic Scheduling-Minimize Individual Makespans of Functions (FDS_MIMF)	At device level	–	High	Moderate	Yes	Yes	Unspecified
Adaptive Dynamic Scheduling-Minimize Individual Makespans of Functions (ADS_MIMF)	At device level	–	High	Moderate	Yes	Yes	Unspecified

- **The need of energy efficiency** – because collaborative cyber-physical systems often contain components which are mobile and, therefore, not powered directly from the electrical network, the need of energy efficiency is crucial. As a result, special scheduling algorithms must be developed in this sense.
- **Heterogeneous hardware** – since collaborative systems can have different architectures and configurations and, on the other hand, in the same collaborative systems there can be various components with different hardware support, a unitary approach regarding MCS implementation can hardly be reached.
- **Different communication protocols** – due to the use of different hardware components and possible layered architectures, different communication protocols can be used between various components (or levels of the system architecture), complicating the system operation.

Since CPS is a broad concept which covers multiple areas, it is important to outline the challenges that may occur with the integration of MCSs. Two main concerns can be emphasized, that should be taken into account:

- Domain specific – some domains lack standardization, which can be applied at the following levels:
  - Task level – generic models for different levels of an application, compatible with various scheduling algorithms.
  - Scheduling level – standardized scheduling algorithms which can be used in all CPS areas.

Using well defined standards provides consistency, quality and improves productivity. This involves a set of rules which should be easily understood, constantly followed and continuously improved.

- Generality – there are no general scheduling mechanisms, most of the approaches are application/system specific. Consequently, solutions to the same problem might vary depending on the application domain. Even though, specifically tailored solutions can provide determinism, predictability, performance and timeliness in certain scenarios, general solutions bring higher flexibility, scalability and interconnectivity between different devices.

These shortcomings can be solved by using a standardized approach applicable to all CPS domains. In this sense, the set of common, constructive and functional features for each area of development previously illustrated should be taken into account. Additionally, using a generic scheduling algorithm is far more

advantageous over application specific methods, both in terms of flexibility and cost.

Solving the previously mentioned challenges will certainly lead to an efficient integration of MCSs in CPSs, with important advantages to this approach on the long run.

5.2. Advantages of integrating MCSs in CPSs

Despite the numerous challenges, incorporating MCSs in CPSs brings some important advantages which are further listed below:

- **Real-time functionalities** – CPSs involve features such as adaptability and safety, which require meeting strict timing constraints when it comes to monitoring the environmental changes, processing data, sharing information and executing certain actions. In some domains such as medical, if a system is not real-time compliant, the consequences can be irreversible. Even though MCSs initially addressed real-time systems, they can also be integrated in systems with both real-time and non-real-time requirements, when different functionalities and components with and without real-time constraints share the same hardware (Mollison et al., 2010).
- **Multiple functionalities on the same platform** – it is advantageous to implement multiple functionalities on the same platform. This means that some components will be more "critical" than others, thus the concept of mixed criticality is slowly coming to be considered in CPSs.
- **Hierarchical scheduling** – a vast majority of the scheduling algorithms are developed at the basic level of an application, i.e. at task level, but in the context of complex distributed systems, where applications from different providers must function together, this approach is insufficient. Additionally, interconnecting scheduling objectives between different computational layers in a complex system ensures transparency and helps meet temporal requirements. Adopting unitary abstraction models for the different layers of an application (task level, meta-task level or system level) may be a huge step forward allowing interconnection of different CPSs regardless of their application field and pushing things closer to standardization between different areas.
- **Standardized approaches** – currently, each CPS domain has its own algorithmic solutions depending on certain requirements and real-time behaviors. This creates inconsistencies when trying to apply the same scheduling technique in different CPS areas. The idea is to reach a unified solution by integrating MCSs in CPSs, which will provide a basis for further development in each domain and better interconnections between critical and non-critical components.
- **Temporal and resource isolation** – important research results in the field of mixed criticality systems have been presented in the last decade, especially in the scheduling theory field. One of the main consequences of this research is that safety-critical systems are no longer restricted to a single type of processing units, fact reflected in the development of a great variety of heterogeneous systems, with different computational and power performances. The temporal and resource isolation mechanism makes possible the use of advanced modern processing units with higher computational capabilities but less predictability in mixed and safety critical systems (Cavalcanti and Freitas, 2005).
- **Adaptability** – another direct implication coming from using the MCS approach is that, by using a model with different parameter values for different running modes, one could make a priori predictions about the evolution of various performance parameters, beside the CPU usage (e.g. run-time energy usage, memory usage, communication bandwidth, etc.) in different running modes (Cavalcanti and Freitas, 2005). This aspect can be of particular interest for the CPS, where the direct environment-system interaction may impose that the system dynamically adapts to different running conditions, depending on variations in the system environment, thus formalizations and methodologies for modeling and off-line testing of the system behavior represent important progress, which must not be neglected.
- **Robustness** – hypervisor techniques and standard distributed middleware can represent general solutions for imposing robustness in terms of run-time behavior, which exceed the fields of automotive and avionics.

The advantages presented above offer new research perspectives in the field.

5.3. Future perspectives

Beside the previously mentioned advantages of integrating MCSs in CPSs, which are more relevant from a practical point of view, new areas of research have been opened:

- Mixed criticality scheduling in IoT systems; The IoT systems are rapidly evolving towards real-time IoT systems (Malik et al., 2019) where tasks with different time constraints and different criticality levels coexist. In these circumstances, a hierarchical mixed criticality scheduling approach, which can be extended to include different IoT scheduling layers such as IoT device scheduling, fog scheduling and cloud service scheduling, becomes a new and significant challenge.
- Fog scheduling – Fog computing (Vaquero and Rodero-Merino, 2014) is an emerging new paradigm where a large number of heterogeneous, ubiquitous and decentralized processing units communicate and cooperate with each other. Thus, beside the need for standardization, integrating MCS approaches in fog computing must meet additional requirements such as: synchronization, security, liability and programmability. This results in the concept of fog scheduling, which is a research subject of high potential.
- Multi-Agent Systems (MAS) mixed criticality task scheduling – a similar set of challenges as the ones presented in our paper have been highlighted by Calvaresi et al. (2017), when it comes to interconnecting MAS to other CPS domains: the lack of standardization, real-time requirements (communication, sharing information, adapting to environmental changes, etc.) and the lack of a task model that includes temporal behavior. Thus, solving some of the issues presented in this paper would have an important impact in the development of other areas like MAS and IoT.
- MCS implementations in CPSs have the potential to bring major improvements in security by integrating the temporal and spatial isolation concept.

From the perspective of the previously mentioned areas of research, new opportunities regarding MCS scheduling algorithms are arising:

- IoT systems – Introducing a mixed criticality approach to IoT would improve the reliability, safety, scalability and transmission efficiency of the systems by extending the message schedulers to handle different criticalities and real-time constraints. Because common web technologies are inefficient for message scheduling when interfacing real-time IoT systems, a distributed heterogeneous approach, like D_MHEFT, F_MHEFT is preferable offering flexibility and adaptability for coordinating multiple components in a dynamic environment.
- Fog scheduling – The same concept is also applicable at the Fog computing level. In this case scheduling algorithms for

both single and multiple processing units can be used, depending on the complexity of the hardware, ranging from single processing unit scheduling algorithms to clustered schedulers. From the single processing unit class, EDF based algorithms such as EDF_VD, CBEDF have a greater potential, being dynamic and energy efficient. Some well-known scheduling methods for multiple processing units that can also be used in Fog computing: P-EDF-VD, P-EKB for partitioned scheduling, MxC-RUN, MC-Fluid for global scheduling and NPS-F-MC, SP-EKB for clustered scheduling.

- Multi-Agent Systems (MAS) – Because MAS is a complex field, the introduction of mixed criticality concepts must be done at different layers. Thus, a hierarchical and decentralized design is suitable, which integrates multiple classes of scheduling algorithms. For example, considering a fleet of robots: a single processing unit scheduling technique based on algorithms like EDF-VD, EDF-DB could be implemented at the component level, partitioned methods such as TTS-MC, MC-PARTITION-UT-INC at the agent level and a distributed approach at the system level.

6. Conclusions

In this paper we have outlined the challenges and potential rewards of integrating MCSs in CPSs. MCSs are expected to play an important role in the design and implementation of future CPSs with capabilities that will exceed today's levels of reliability, security and functionality. Such an approach has a high potential as it allows multiple functionalities on the same platform, thus different components can be interconnected for safe and adaptable systems.

Even though there is no scheduling algorithm suitable for all the cyber physical systems application fields, some algorithms are more appropriate considering certain attributes. As future work, we will focus on finding different classes of applications and validate suitable algorithms for each class.

A topic of future work is to move away from an application specific approach and to focus on more general, standardized scheduling techniques. The standardization of the application and scheduling techniques will ease the development of different operating systems, of different distributed applications interconnecting wide areas like IoT, automotive, medical and even military into single complex systems running complex applications, like real-time surveillance and image recognition, remote control for different system parts like robots, actuators etc.

References

Accettura, N., Palattella, M.R., Boggia, G., Grieco, L.A., Dohler, M., 2013. Decentralized traffic aware scheduling for multi-hop low power lossy networks in the internet of things. In: World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a. IEEE, pp. 1–6.

Ali, A., Kim, K.H., 2017. Cluster-based multicore real-time mixed-criticality scheduling. J. Syst. Archit. 79, 45–58.

Ali, I., Seo, J.H., Kim, K.H., 2015. A dynamic power-aware scheduling of mixed-criticality real-time systems. In: 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing. IEEE, pp. 438–445.

Arlock, C.C., Linderoth-Olson, E., 2014. A practical comparison of scheduling algorithms for mixed criticality embedded systems. Department of Automatic Control, Lund University, Sweden, pp. 1–48.

Asyaban, S., Kargahi, M., 2017. An exact schedulability test for fixed-priority preemptive mixed-criticality real-time systems. Real-Time Syst. 1–59.

Awan, M.A., Bletsas, K., Souto, P.F., Tovar, E., 2017. Semi-partitioned mixed-criticality scheduling. In: International Conference on Architecture of Computing Systems. Springer, Cham, pp. 205–218.

Baruah, S., Guo, Z., 2015. Mixed-criticality job models: a comparison. In: Proceedings of the Workshop on Mixed-Criticality Systems (WMC'15).

Baruah, S.K., Burns, A., Davis, R.I., 2011a. Response-time analysis for mixed criticality systems. In: Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd. IEEE, pp. 34–43.

Baruah, S.K., Burns, A., Davis, R.I., 2011b. Response-time analysis for mixed criticality systems. In: 2011 IEEE 32nd Real-Time Systems Symposium. IEEE, pp. 34–43.

Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Spaccamela, A., M., 2012. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In: 24th Euromicro Conference on Real-Time Systems (ECRTS12), Jul 2012, Pisa, Italy. IEEE, pp. 145–154.

Baruah, S., Chattopadhyay, B., Li, H., Shin, I., 2014. Mixed-criticality scheduling on multiprocessors. Real-Time Syst. 50 (1), 142–177.

Baruah, S.K., 2004. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. IEEE Trans. Comput. 53 (6), 781–784.

Braun, T.D., Siegal, H.J., Beck, N., Boloni, L.L., Maheswaran, M., Reuther, A.I., Freund, R.F., 1999. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In: Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth. IEEE, pp. 15–29.

Burke, E.K., Hyde, M.R., Kendall, G., 2006. Evolving bin packing heuristics with genetic programming. In: Parallel Problem Solving from Nature-PPSN IX. Springer, Berlin, Heidelberg, pp. 860–869.

Burns, A., Davis, R.I., 2017. A survey of research into mixed criticality systems. ACM Comput. Surv. (CSUR) 50 (6), 82.

Burns, A., 2015. An augmented model for mixed criticality. In: Baruah, S.K., Cucu–Grosjean, L., Davis, R.I., Maiza, C. (Eds.). Mixed Criticality on Multicore/Manycore Platforms (Dagstuhl Seminar 15121), 5, editors. SchlossDagstuhl–Leibniz-ZentrumfuerInformatik, Dagstuhl, Germany.

Calandrino, J.M., Leontyev, H., Block, A., Devi, U.C., Anderson, J.H., 2006. LITMUS^ RT: a testbed for empirically comparing real-time multiprocessor schedulers. In: Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International. IEEE, pp. 111–126.

Calvaresi, D., Marinoni, M., Sturm, A., Schumacher, M., Buttazzo, G., 2017. The challenge of real-time multi-agent systems for enabling IoT and CPS. In: Proceedings of the International Conference on Web Intelligence. ACM, pp. 356–364.

Capota, E.A., Stangaciu, C.S., Micea, M.V., Cretu, V.I., 2018. P_FENP: a multiprocessor real-time scheduling algorithm. In: 2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI), Timisoara, pp. 509–514.

Carpenter, T., Hatcliff, J., Vasserman, E.Y., 2017. A reference separation architecture for mixed-criticality medical and IoT devices. In: Proceedings of the 1st ACM Workshop on the Internet of Safe Things. ACM, pp. 14–19.

Cavalcanti, A., Freitas, R.A., 2005. Nanorobotics control design: a collective behavior approach for medicine. IEEE Trans. Nanobiosci. 4 (2), 133–140.

Crespo, A., Ripoll, I., Masmano, M., Peiró, S., 2010. Partitioned embedded architecture based on hypervisor: the XtratuM approach. In: European Dependable Computing Conference (EDCC), pp. 67–72.

Crespo, A., Alonso, A., Marcos, M., Juan, A., Balbastre, P., 2014. Mixed criticality in control systems. IFAC Proc. Vol. 47 (3), 12261–12271.

De Niz, D., Lakshmanan, K., Rajkumar, R., 2009. On the scheduling of mixed-criticality real-time task sets. In: Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE. IEEE, pp. 291–300.

Ekberg, P., Yi, W., 2014. Bounding and shaping the demand of generalized mixed–criticality sporadic task systems. Real-Time Syst. 50 (1), 48–86.

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

Fakih, M., Lenz, A., Azkarate-Askasua, M., Coronel, J., Crespo, A., Davidmann, S., Seyyedi, R., 2017. SAFEPOWER project: architecture for safe and power-efficient mixed-criticality systems. Microprocess. Microsyst. 52, 89–105.

Funk, S., Levin, G., Sadowski, C., Pye, I., Brandt, S., 2011. DP-fair: a unifying theory for optimal hard real-time multiprocessor scheduling. Real-Time Syst. 47 (5), 389.

Gerostathopoulos, I., Bures, T., Hnetynka, P., Keznikl, J., Kit, M., Plasil, F., Plouzeau, N., 2016. Self-adaptation in software-intensive cyber–physical systems: from system goals to architecture configurations. J. Syst. Softw. 122, 378–397.

Giannopoulou, G., Huang, P., Ahmed, R., Bartolini, D.B., Thiele, L., 2017. Isolation scheduling on multicores: model and scheduling approaches. Real-Time Syst. 53 (4), 614–667.

Giannopoulou, G., 2017. Implementation of Mixed-Criticality Applications on Multi–Core Architectures. ETH Zurich.

Gratia, R., Robert, T., Pautet, L., 2015. Scheduling of mixed-criticality systems with RUN. In: 2015 IEEE 20th Conferenceon Emerging Technologies & Factory Automation (ETFA), 2015, pp. 1–8.

Guan, N., Ekberg, P., Stigge, M., Yi, W., 2011. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In: RTSS, pp. 13–23.

Han, J.-J., Tao, X., Zhu, D., Aydin, H., Shao, Z., Yang, L., 2018. Multicore mixed-criticality systems: partitioned scheduling and utilization bound. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 37 (1) Jan.

Huang, P., Kumar, P., Giannopoulou, G., Thiele, L., 2014. Energy efficient dvfs scheduling for mixed-criticality systems. In: Proceedings of the 14th International Conference on Embedded Software. ACM, p. 11.

Huang, P., Giannopoulou, G., Ahmed, R., Bartolini, D.B., Thiele, L., 2015. An isolation scheduling model for multicores. In: 2015 IEEE Real-Time Systems Symposium. IEEE, pp. 141–152.

Kamienski, C., Jentsch, M., Eisenhauer, M., Kiljander, J., Ferrera, E., Rosengren, P., Sadok, D., 2017. Application development for the internet of things: a context-aware mixed criticality systems development platform. Comput. Commun. 104, 1–16.

Lakshmanan, K., De Niz, D., Rajkumar, R., Moreno, G., 2010. Resource allocation in distributed mixed-criticality cyber-physical systems. In: Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on. IEEE, pp. 169–178.
Lakshmanan, K., de Niz, D., Rajkumar, R., 2011. Mixed-criticality task synchronization in zero-slack scheduling. In: 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium. IEEE, pp. 47–56.
Laplante, P.A., 2004. Real-Time Systems Design and Analysis . Wiley, New York p. Xxi.
Larrucea, A., Martinez, I., Brocal, V., Peirò, S., Ahmadian, H., Perez, J., Obermaisser, R., 2016. DREAMS: cross-domain mixed-criticality patterns. In: Workshop on Mixed-Criticality Systems, p. 6.
Lee, J., Shin, K.G., 2017. Development and use of a new task model for cyber-physical systems: a real-time scheduling perspective. J. Syst. Softw. 126, 45–56.
Lee, J., Phan, K.-M., Gu, X., Lee, J., Easwaran, A., Shin, I., Lee, I., 2014. MC-fluid: fluid model-based mixed-criticality scheduling on multiprocessors. In: RTSS, pp. 41–52.
Lee, J., Chwa, H., Phan, S., Shin, L., T.X., Lee, I., 2017. MC-ADAPT: adaptive task dropping in mixed-criticality scheduling. International Conference on Embedded Software (EMSOFT 2017), 16 163:1–163:21October.
Lee, E.A., 2008. Cyber physical systems: design challenges. In: 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC). IEEE, pp. 363–369.
Legout, V., Jan, M., Pautet, L., 2013. Mixed-criticality multiprocessor real-time systems: energy consumption vs deadline misses. In: First Workshop on Real-Time Mixed Criticality Systems (ReTiMiCS), pp. 1–6.
Lu, C., 2016. Mixed-Criticality Scheduling of an Autonomous Driving Car. Master's Thesis in Institute for Integrated Systems. Technische Universität München, Department of Electrical and Computer Engineering.
Müller, D., Werner, M., 2011. Genealogy of hard real-time preemptive scheduling algorithms for identical multiprocessors. Open Comput. Sci. 1 (3), 253–265.
Malik, S., Ahmad, S., Ullah, I., Park, D.H., Kim, D., 2019. An adaptive emergency first intelligent scheduling algorithm for efficient task management and scheduling in hybrid of hard real-time and soft real-time embedded IoT systems. Sustainability 11 (8), 2192.
Micea, M.V., Stangaciu, C.S., Stangaciu, V., Curiac, D.I., 2017. Novel hybrid scheduling technique for sensor nodes with mixed criticality tasks. Sensors 17 (7), 1504.
Mollison, M.S., Erickson, J.P., Anderson, J.H., Baruah, S.K., Scoredos, J.A., 2010. Mixed-criticality real-time scheduling for multicore systems. In: 2010 10th IEEE international conference on computer and information technology. IEEE, pp. 1864–1871.
Narayana, S., Huang, P., Giannopoulou, G., Thiele, L., Prasad, R.V., 2016. Exploring energy saving for mixed-criticality systems on multi-cores. In: 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE, pp. 1–12.
Object Management Group, 2007. Data distribution service for real-time SystemsOMG document, v1.2, formal/07-01-01.
Pérez, H., Gutiérrez, J.J., Peiró, S., Crespo, A., 2017. Distributed architecture for developing mixed-criticality systems in multi-core platforms. J. Syst. Softw. 123, 145–159.
Park, T., Kim, S., 2011. Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems. In: Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on. IEEE, pp. 253–262.
Ren, J., Phan, L., T., X., 2015. Mixed-criticality scheduling on multiprocessors using task grouping. In: 2015 27th Euromicro Conference on Real-Time Systems, 2015, pp. 25–34.
Rodriguez, P., George, L., Abdeddaïm, Y., Goossens, J., 2013. Multicriteria evaluation of partitioned EDF-VD for mixed-criticality systems upon identical processors. Workshop on Mixed Criticality Systems.
Santy, F., George, L., Thierry, P., Goossens, J., 2012. Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp. In: Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on. IEEE, pp. 155–165.
Schneider, R., Goswami, D., Masrur, A., Becker, M., Chakraborty, S., 2013. Multi-layered scheduling of mixed-criticality cyber-physical systems. J. Syst. Archit. 59 (10), 1215–1230.
Stigge, M., Yi, W., 2015. Graph-based models for real-time workload: a survey. Real-Time Syst. 51 (5), 602–636.
Taherin, A., Salehi, M., Ejlali, A., 2018. Reliability-aware energy management in mixed-criticality systems. IEEE Trans. Sustain. Comput. 3 (3), 195–208.
Völp, M., Hähnel, M., Lackorzynski, A., 2014. Has energy surpassed timeliness? Scheduling energy-constrained mixed-criticality systems. In: 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE, pp. 275–284.
Vaquero, L.M., Rodero-Merino, L., 2014. Finding your way in the fog: towards a comprehensive definition of fog computing. ACM SIGCOMM Comput. Commun. Rev. 44 (5), 27–32.
Vestal, S., 2007. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In: Proceedings of the Real-Time Systems Symposium. IEEE Computer Society Press, Tucson, AZ, pp. 239–243. December.
Xia, C., Jin, X., Kong, L., Wang, J., Zeng, P., 2017. Transmission scheduling for mixed–critical multi-user multiple-input and multiple-output industrial cyber-physical systems. Int. J. Distrib. Sens. Netw. 13 (12), 1–13.
Xie, G., Zeng, G., Liu, L., Li, R., Li, K., 2016. High performance real-time scheduling of multiple mixed-criticality functions in heterogeneous distributed embedded systems. J. Syst. Archit. 70, 3–14.
Xie, G., Zeng, G., Li, Z., Li, R., Li, K., 2017. Adaptive dynamic scheduling on multifunctional mixed-criticality automotive cyber-physical systems. IEEE Trans. Veh. Technol. 66 (8), 6676–6692.
Zhan, J., Zhang, X., Jiang, W., Ma, Y., Jiang, K., 2018. Energy optimization of security-sensitive mixed-criticality applications for distributed real-time systems. J. Parallel Distrib. Comput. 117, 115–126.
Zhao, Q., Gu, Z., Zeng, H., Zheng, N., 2018. Schedulability analysis and stack size minimization with preemption thresholds and mixed-criticality scheduling. J. Syst. Archit. 83, 57–74.
Zhou, S., Zheng, X., Wang, J., Delisle, P., 1993. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. Software 23 (12), 1305–1336.

Eugenia Ana Capota is a Ph.D. student in Computer and Information Technology at the Politehnica University of Timisoara, Romania. Her current research interests include cyber-physical systems, real-time task scheduling and schedulability analysis of mixed-criticality systems.

Cristina Sorina Stângaciu (born Cristina Sorina Certejan) is a lecturer and a research engineer at the Department of Computer and Information Technology, Politehnica University of Timisoara, Romania. Her research areas and interests include: embedded and real-time hardware-software systems; power management in embedded devices, real-time scheduling techniques.

Mihai Victor Micea is a professor and Chair of the Department of Computer and Information Technology, Politehnica University of Timisoara, Romania. His research interests include real-time/embedded multiprocessing systems with applications in DAQ and DSP, robotic collectives, intelligent sensor networks and multimedia systems.

Daniel-Ioan Curiac is a professor in the Department of Automation and Applied Informatics, Politehnica University of Timisoara, Romania. His current research interests include cyber physical systems, robotic systems, real-time adaptive systems and information security.