Bridging the Pragmatic Gaps for Mixed-Criticality Systems in the Automotive Industry

Zhe Jiang*‡, Shuai Zhao†, Ran Wei§, Richard Parterson‡, Nan Guan††, Yan Zhuang§, Neil Audsley¶,
*University of Cambridge, United Kingdom, ‡ARM Ltd., United Kingdom,
†University of York, United Kingdom, ¶City, University of London, United Kingdom,
§Dalian University of Technology, China, ††City, University of Hong Kong, China

I. Introduction

There is an increasing trend for safety-critical systems to be integrated onto a shared platform to achieve functions of different criticalities [1] [2]. Such systems are often referred to as *Mixed-Criticality Systems (MCS)s* [3]. With the ever-increasing demand of system functionalities and the shift of the semiconductor industry to more powerful (multi-core) platforms, mixing functionalities of different criticality levels in a common hosting platform is appealing - system costs introduced by size, weight and power consumption could potentially be significantly reduced in the mixed-criticality setting [4]. For example, leading automated vehicle companies are devoting themselves to integrating Electronic Control Unit (ECU) clusters and In-Vehicle Information System (IVIS) in a shared platform. In such system, the ECU clusters must be developed with the highest criticality as they usually control the mechanical components (e.g. engine control), whereas the IVIS can be developed with relatively lower criticality as it only provides the interactive functionalities (e.g. navigation).

In academia, extensive research efforts have been made towards MCS [3]. However, in industries, limited development guidance for MCS have been provided in industrial standards (e.g. DO-178C for avionics, ISO 26262 for automotive and EN 50128 for railway). Consequently, no standard industrial system architecture for MCS has been established [5]. This is due to the fact that a pragmatic gap exists between theoretical MCS models and industrial practice – researchers have not taken industrial practice/requirements into sufficient consideration, causing conceptual mismatches to emerge between theoretical models and industrial architectures. This is observed by a number of studies, in [6], multiple definitions on *criticality* between the theoretical MCS models and industrial standards are discussed; in [7], the applicability of *graceful degradation* from theoretical MCS models to the industrial context is discussed. To the best of our knowledge, no systematic development methodology for MCS in industrial scenarios has been proposed.

Contributions. In [8], we proposed a new system architecture to take the first practical step in an attempt to bridge the gaps between theoretical MCS models and industrial practices. The original contributions made in [8] include i) a *systematic approach* to develop MCS in an industrial scenario, which is proposed upon Adaptive Mixed-Criticality (AMC) MCS model [3] with considerations of industrial practice and requirements; ii) a *system architecture* (*P-MCS*) and three *design methodologies* for the proposed approach; iii) a *theoretical model* and *schedulability analysis* for *P-MCS*, which guarantees system predictability; and iv) comprehensive *experiments* to examine *P-MCS* against conventional MCS frameworks in different perspectives.

This paper makes the following additional contributions: i) a comprehensive *analysis* of pragmatic gaps between MCS theory in academia and industrial practice; ii) a *systematic approach* to develop MCS in an industrial scenario, which is proposed upon Adaptive Mixed-Criticality (AMC) MCS model [3] with considerations of industrial practice and requirements; iii) a *theoretical model* and *schedulability analysis* for *P-MCS*, which guarantees system predictability; and iv) comprehensive *experiments* to examine *P-MCS* against conventional MCS frameworks in different perspectives.

II. The Academic Model

The majority of the academic research effort on MCS relies on the AMC model proposed by Vestal [3]. The AMC model assumes that the system has several execution modes ($L \in \{(Mode)\ A, B, C, D, ...\}$), and contains a finite set of *sporadic tasks*. Each task τ_i is defined by its period (T_i), relative deadline (D_i), a priority P_i, a criticality level ($l_i \in \{A, B, C, D, ...\}$), and a set of Worst-Case Execution Time (WCET) estimations ($\{C_{i,A}, C_{i,B}, ..., C_{i,l_i}\}$), in which these estimations reflect the WCET of τ_i in the criticality level, up to l_i. The model assumes that $C_{i,A} \leq C_{i,B} \leq ...C_{i,l_i}$. Specifically, the measured WCET at the lowest system mode (i.e. $L = A$) is set to $C_{i,A}$, whereas at each higher system mode, the subsequent estimations ($C_{i,B}, ..., C_{i,l_i}$) are obtained by either more pessimistic WCET analysis techniques, or by considering safety margins imposed by certification authorities. The system initialises from system mode A, and all tasks are scheduled to execute. During execution, if any task τ_i exceeds its execution budget ($C_{i,A}$), the system will switch to the next mode (i.e. $L = B$). In the meantime, tasks with a criticality level lower than B ($l_i < B$) are suspended.

In the first AMC work [3], only a single-core MCS with two system modes (i.e. Low- and High-criticality modes) is considered. The AMC model is further extended by various works, e.g. extensions on multiple system modes and criticality levels [9], re-activation of the dropped tasks [10], etc. (see [2] for a comprehensive survey). Our proposed *P-MCS* architecture is also built atop the AMC model, with assumptions that the system does not switch modes backwards, and tasks do not reactivate after being terminated/suspended.

*A full version of this paper was published in the IEEE IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) [1].

[1]By criticality, we mean the required safety assurance level of system components, such as the Automotive Safety Integrity Level (ASIL) from the international automotive safety standard ISO 26262.

III. The Industrial Gaps

Our work is motivated by two observations obtained from analysing the pragmatic gap between the AMC model and practice in the automotive industry.

A. The Breaking of the ASIL Allocation System

The core concept of the AMC model is *system mode switch*, which guarantees the execution of tasks with higher criticality, by suspending tasks with lower criticality [2], [3]. As summarised by Burns and Davis [2], the majority of the research links the *system mode switch* to '*graceful degradation*' from industrial standards, i.e. '*a technique aimed at maintaining the more important system functions available, despite failures, by dropping the less important system functions*' (ISO 26262-4:2018, Clause 7). Therefore, determining 'more important functions' and suspending 'less important functions' is the key of implement *system mode switch* correctly in an industrial MCS architecture.

However, ISO 26262 provides a means to determine the criticality of the requirement of a function, which considers *S* (Severity), *E* (Exposure) and *C* (Controllability). According to the philosophy of *system mode switch* of the AMC model, tasks assigned with higher *ASIL* requirements must always be allowed to execute (when tasks assigned with lower *ASIL* cannot finish execution within the time given to them), even if it means that tasks assigned with lower *ASIL* requirements will be suspended. This means, for example, when the AMC model executes at system mode C ($L = C$), which enables *ASIL-C* and *ASIL-D* tasks to execute, it will suspend all *ASIL-A* and *ASIL-B* tasks, regardless of their associated *S*, *E* and *C* classes. This is problematic, consider an *ASIL-B* task τ_i ($l_i = B$) with $\{$*S3*, *E3* and *C2*$\}$ and an *ASIL-C* task τ_j ($l_j = C$) with $\{$*S2*, *E4* and *C3*$\}$. When AMC determines that task τ_j needs to execute and perform mode switch from mode B ($L = B$) to mode C ($L = C$), task τ_i is suspended. However, what has not been considered is that the *Exposure* of task τ_i is raised from *E3* to *E4* because its intended function will always fail (because it is suspended by the mode switch). In addition, failure of task τ_i causes more harm than τ_j for it has an *S3* Severity class. As previously discussed, ASILs are statically allocated, this means that the safety requirement of τ_i cannot be raised at run-time, even though the suspension of τ_i essentially raises its safety requirement from *ASIL-B* ($l_i = B$) to *ASIL-C* ($l_i = C$).

Another problem caused by the *system mode switch* is the suspension of tasks with *ASIL* allocated to their requirements as a result of ***ASIL Decomposition***. For example, provided a safety requirement $R1$ with *ASIL-D* is decomposed into $R1.1$ (*ASIL-C*) and $R2.2$ (*ASIL-A*), where τ_i (with $l_i = C$) and τ_j (with $l_j = A$) are tasks fulfilling these two safety requirements. When the AMC model switches from Mode A ($L = A$) to Mode B ($L = B$), task τ_j is suspended as $l_j < L$. Although τ_j is independent from τ_i (ISO 26262 enforces component independence when performing ASIL Decomposition), the suspension of τ_j poses threats to the fulfilment of safety requirement $R1$. Another more severe problem is when $R1$ is decomposed into $R1.1$ (*ASIL-B*) and $R2.2$ (*ASIL-B*), that both tasks fulfilling $R1.1$ and $R1.2$ will be suspended when the AMC switches from Mode B ($L = B$) to Mode C ($L = C$). The above observations give rise to the first pragmatic gap of AMC:

Pragmatic Gap I. *During the system mode switch, safety analysis must be performed (either run-time or off-line) on the executing tasks to determine ones that need to be preserved, in order to avoid catastrophic consequences caused due to the termination of these applications.*

B. Isolation v.s. Freedom from Interference

In safety-related standards (including ISO 26262), isolation/separation between different critical functions is presented as the essential requirement for MCS. As regulated in ISO 26262, '*If freedom from interference between elements implementing safety requirements cannot be argued in the preliminary architecture, then the architectural elements shall be developed in accordance with the highest ASIL for those safety requirements.*'. This implies that without certain isolation, all elements have to be treated with the highest criticality. Although *ASIL Decomposition* can be applied, in practice, *ASIL Decomposition* is rather abused without considering isolation/separation [11].

Isolation must be sufficiently considered in these dimensions: temporal isolation, spatial isolation and fault isolation. In an industrial MCS architecture, different criticality levels introduce diverse requirements in the design and verification, which lead to different fault-tolerance capabilities among different critical applications (i.e. faults have more possibility occurring in a low-criticality application, compared to a high-criticality application [5], [7]). Spatial and fault isolation avoid fault propagation between applications with different criticality levels. Whilst temporal isolation effectively avoids the propagation of malfunctions, e.g. caused by consuming too high processor execution time (performance isolation) [2], [7]. This leads to the second Pragmatic Gap:

Pragmatic Gap II. *In the industrial MCS architecture, temporal isolation, spatial isolation and fault isolation must be guaranteed between the different critical elements.*

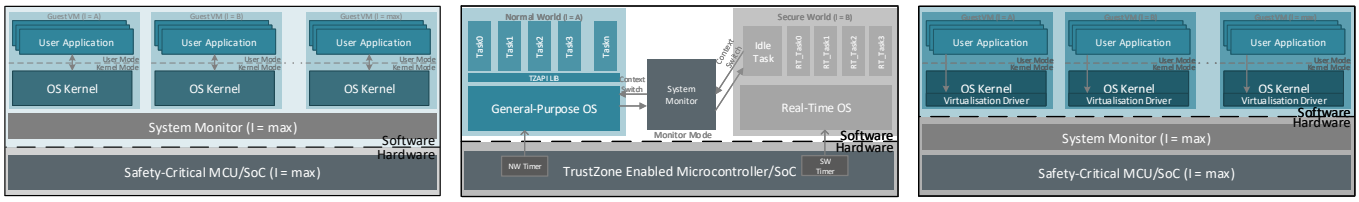
Although application-level isolation between different critical components has already been introduced in existing MCS frameworks, the required isolation must consider the entire system architecture, including Operating System (OS), system monitor, device drivers, and hardware platform. Detailed discussion regarding partitioning and isolation is provided in [1].

IV. Practical Mixed-Criticality System Architecture (P-MCS)

With sufficient consideration of the AMC model and industrial requirements, we propose a generic industrial MCS architecture, termed *P-MCS* (Practical-Mixed-Criticality System) to bridge the pragmatic gaps identified. The *P-MCS* inherits most of its features from the AMC model (e.g. system mode switch) with the additional consideration of industrial requirements. The proposed architecture is suitable for generic and sightly forward-looking MCS industrial scenarios, e.g. systems with more than four criticality levels.

A. Run-time Safety Analysis

To determine the correct 'important functions' in *system mode switch* (Pragmatic Gap I), a run-time two-level safety analysis is proposed to examine each task at the current system mode, and to determine the *preserved* task set and the *terminated* task set for the system mode to be switched into.



(a) Software Virtualisation (b) TrustZone-based Implementation (c) Hardware Acceleration

Fig. 1. Implementation Examples of *P-MCS* (Two Criticality Levels Supported)

A *preserved* task in mode K indicates the task is allowed to execute in mode K but with a lower criticality level. A *terminated* task in mode K is a task that will be terminated during the mode switch to K.

Level 1: Failure Modes and Effect Analysis (FMEA[2]). At the first level, the impacts of terminating each task are analysed. If the termination of the task causes an unacceptable consequence in the next system mode, the task is preserved. Otherwise, the task is added into the termination list for the time being and is passed to the level 2 analysis.

Level 2: Dependency Analysis. At this level, the dependency of the important tasks (output from level 1) is analysed. Any task that can cause corruption of an important task due to its termination, has to be kept in the next system mode.

B. Support for Isolation

Considering isolation (Pragmatic Gap II), applications with different criticality levels are allocated in the independent executing environment (including spatial, temporal and fault isolation). The mapping from isolated application groups to operating systems is performed off-line by allocation algorithms (e.g. Worst-Fit Decreasing). The corresponding scheduling between the isolated environment is supported by the system monitor (more privileged). Because the hardware platform and low-level drivers can be simultaneously accessed by different critical applications without '*freedom from interference*', both hardware platforms[3] and shared low-level drivers are designed and executed at the highest criticality level. Following the same rationale, the system monitor is also executed at the highest criticality level. Differently, with independent high-level drivers or operating systems being provided to the different critical applications (e.g. kernel separation), such high-level drivers or OSs only need to inherit the highest criticality from accessing applications. Notably, some low-level drivers are not accessed by all components [4]. These low-level drivers only need to inherit the highest criticality of the accessed applications.

C. Implementing P-MCS

We introduce three possible ways to implement *P-MCS*.

1) Software Virtualisation ($P|swv$): Virtualisation technology enables spatial, temporal, and fault isolation between guest VMs [12]. In order to achieve isolated executing environment for applications with different criticality, $P|swv$ assigns different criticality levels to guest VMs and allocates the applications correspondingly. During *system mode switch*, if all tasks in a guest VM are suspended, the guest VM can be terminated directly. $P|swv$ is agnostic to the applied virtualisation. In our implementation, we adopt Xen [13]. The system architecture of $P|swv$ is shown in Fig. 1(a).

System Monitor. In $P|swv$, system monitor is integrated into the ready-built Virtual Machine Monitor (VMM), e.g. Xen hypervisor [13], which is mainly responsible for: virtualisation (e.g. interposition), the run-time safety analysis, system mode switch (including execution monitoring), and real-time scheduling of the VMs.

2) Trusted-Execution Environment ($P|tz$): ARM TrustZone is a hardware-assisted separation technology introduced in Cortex-A processors since 2004 [14]. This technology is centred around the concept of separating the system execution into two independent executing environments (i.e. *secure world* and *normal world*). Such worlds are granted uneven privileges, e.g. normal software is prevented from directly accessing secure world resources. For isolation, critical and less-critical applications are allocated to the secure and the normal world, respectively. The system architecture is presented in Fig. 1(b). The key limitation of this implementation is that only two criticality levels can be supported on each processor.

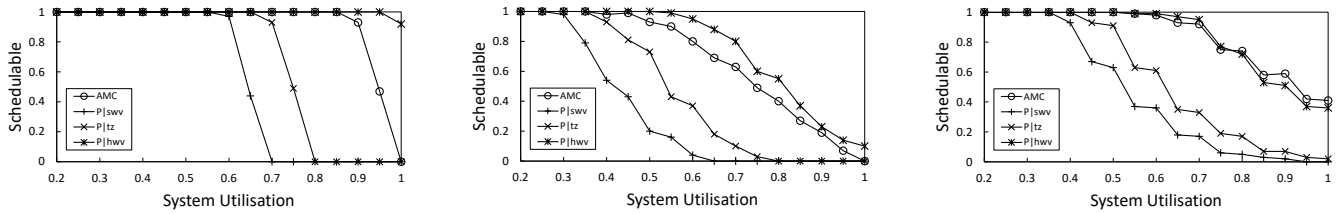
System Monitor. In $P|tz$, the system monitor is implemented in the additional privileged mode (i.e. monitor mode), and is mainly responsible for: inter-domain context switches, run-time safety analysis, and system mode switch (including execution monitoring).

3) Hardware-assisted Virtualisation ($P|hwv$): In *P-MCS*, the additional features introduced in the system monitor and the isolation imposed between different critical domains lead to extra overhead compared to the conventional theoretical model. This overhead significantly undermines the system performance and predictability. $P|hwv$ proposes the same design concept as $P|swv$, but implements the system monitor based on an open-source hardware-designed VMM [15]. The hardware-designed VMM effectively offloads the previously introduced overhead and low-layer drivers from the software to hardware, which simplifies the access paths between the VMs and the underlying hardware. The methodology effectively improves system performance and schedulability, compared to previous implementations. The system architecture of $P|hwv$ is presented in Fig. 1(c). The key limitation of the implementation is the additional hardware overhead and modification of OS kernels (to support hardware-designed hypervisor).

System Monitor. In $P|hwv$, the system monitor is integrated into the hardware-designed VMM, which is mainly responsible for: the majority of virtualisation, run-time safety analysis,

[2]The details of implementing FMEA can be found in any textbook of safety.

[3]Like the software, criticality levels must also be assigned to the hardware components (out-of-scope of this paper, see [4]).



(a) low mode (b) mode switch (c) high mode

Fig. 2. System schedulability under each execution mode

system mode switch (including execution monitoring), and real-time scheduling of the VMs.

V. Response Time Analysis

With the system architecture and run-time behaviours described in Sec. IV, the response time analysis is developed in this section to provide the timing bound for the *P-MCS*. Different from the response time analysis for the traditional dual-mode (high and low) AMC architecture [16], the analysis proposed along with *P-MCS* is generic (applicable to systems with two or more modes), with additions to reflect the unique features of the proposed architecture (e.g. run-time safety analysis and task preservation). We acknowledge that a tighter response time bounding during a mode change is presented in [16]. For details of the analysis, please see [1].

VI. Evaluation

To compare the resulting schedulability of *P-MCS* with the traditional AMC model, the typical uni-processor dual-mode MCS system is used for evaluation. The utilisation of each task is generated by the UUniFast-Discard algorithm, with a total system utilisation bound given by $0.05 \times |\Gamma|$. Task utilisation is computed for the low-mode. Periods are generated in a log-uniform distribution between $[1ms, 1000ms]$, with implicat deadlines. Priorities are given by the deadline-monotonic policy. Among all generated tasks, half of the tasks are randomly chosen to assign with the low criticality level, with others set to high-criticality tasks. The system has 4 severity levels with $S_{low} = 1$ and $S_{high} = 3$. For each task, its severity is generated randomly in a uniform distribution between $[1, 4]$. In addition, task dependency under *P-MCS* is considered and is generated randomly, in which each high-criticality task has 10% possibility to have a dependent low-criticality task. 10,000 systems are performed for each test configuration.

Results. Fig. 2 presents the percentage of schedulable systems of the AMC model (tested by the analysis in [16]) and *P-MCS* (tested by the analysis proposed in Sec. V) under each system execution state with shared resources. The system utilisation is incremented by 5%. With the system running in the low mode (Fig. 2(a)), the AMC model outperforms $P|swv$ due to the additional facilitiesin the *P-MCS*. For $P|tz$, the schedulability loss is reduced by implementing the system monitor on hardware. Notably, $P|hmv$ at the low mode outperforms the AMC model even with the additional costs, via the hardware acceleration. Similar observations are also obtained during the mode switch (Fig. 2(b)). However, the schedulability difference between $P|hwv$ and AMC under the mode switch becomes less significant due to the execution of safety analysis and the potential increased interference from the system monitor. At the high mode (Fig. 2(c)), the schedulability of AMC and $P|hmv$ becomes similar (with AMC slightly better) for preserving additional low-criticality tasks, where more tasks will be preserved under *P-MCS* with the increase of utilisation.

VII. Conclusion

In this paper, we formalise and analyse the mismatches between the MCS theoretical models and industrial standards via the system architecture perspective. Then, we present a generic industrial architecture (i.e. *P-MCS*) upon the conventional AMC, with additional satisfaction on the industrial safety requirements – i.e. run-time safety analysis and isolation between different critical elements. Experimental results show that the hardware-based implementation of P-MCS effectively alleviates the additional cost for satisfying the industrial safety requirements and outperforms the traditional AMC model.

For more details regarding the paper, please see [1].

References

[1] Z. Jiang, S. Zhao, R. Wei, D. Yang, R. Paterson, N. Guan, Y. Zhuang, and N. C. Audsley, "Bridging the pragmatic gaps for mixed-criticality systems in the automotive industry," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 1116–1129, 2021.

[2] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, 2013.

[3] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *RTSS*, 2007.

[4] "26262 road vehicles-functional safety," *International Standard ISO*, 2018.

[5] A. Esper, G. Nelissen, V. Nélis, and E. Tovar, "An industrial view on the common academic understanding of mixed-criticality systems," *Real-Time Systems*, 2018.

[6] P. Graydon and I. Bate, "Safety assurance driven problem formulation for mixed-criticality scheduling," *Proc. WMC, Real-Time Systems Symposium*, pp. 19–24, 2013.

[7] R. Ernst and M. Di Natale, "Mixed criticality systems—a history of misconceptions?" *IEEE Design & Test*, 2016.

[8] Z. Jiang, S. Zhao, P. Dong, Y. Dawei, R. Wei, N. Guan, and N. Audsley, "Re-thinking mixed-criticality architecture for automotive industry," in *38th International Conference on Computer Design*, 2020.

[9] N. Kim, S. Tang, N. Otterness, J. H. Anderson, F. D. Smith, and D. E. Porter, "Supporting I/O and IPC via fine-grained OS isolation for mixed-criticality RT tasks," in *RTNS*, 2018.

[10] S. Baruah and A. Burns, "Implementing mixed criticality systems in ada," in *International Conference on Reliable Software*, 2011.

[11] D. D. Ward and S. E. Crozier, "The uses and abuses of asil decomposition in iso 26262," in *7th IET International Conference on System Safety, incorporating the Cyber Security Conference*, 2012.

[12] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[13] X. Website, "https://xenproject.org/," 2020.

[14] S. Pinto and N. Santos, "Demystifying trustzone: A comprehensive survey," *ACM Computing Surveys (CSUR)*, 2019.

[15] Z. Jiang, N. Audsley, and P. Dong, "Bluevisor: A scalable real-time hardware hypervisor for many-core embedded systems," in *RTAS*, 2018.

[16] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *Real-Time Systems Symposium*, 2011.