Enabling Software Technologies for Critical COTS-based Spacecraft Systems

Invited Paper

Federico Reghenzani
federico.reghenzani@polimi.it
Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy
ESTEC, European Space Agency, Noordwijk, Netherlands

ABSTRACT

In this position article, we motivate the necessity to introduce three software methods in spacecraft computing platforms in order to enable to use COTS components: SIHFT, mixed-criticality, and probabilistic timing analysis. We investigate the benefits and the drawbacks of these techniques, especially in terms of safety, by also analyzing the standards to identify the current limitations that do not allow such techniques to be used. Finally, we recap current and future works, highlighting possible changes to standards.

CCS CONCEPTS

• **Applied computing** → *Avionics*; • **Computer systems organization** → **Real-time systems**.

KEYWORDS

Safety-Critical Systems, SIHFT, Mixed-Criticality, Real-Time

ACM Reference Format:
Federico Reghenzani. 2023. Enabling Software Technologies for Critical COTS-based Spacecraft Systems: Invited Paper. In *20th ACM International Conference on Computing Frontiers (CF '23), May 9–11, 2023, Bologna, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3587135.3592765

1 CHALLENGES IN COTS-BASED CRITICAL SYSTEMS

The design and implementation expenses of nowadays unmanned spacecraft and satellites are dominated by electronics, which is a difficult-to-predict and, often, underestimated cost [17]. The increasing demand for computational power by modern applications requires complex computational platforms, such as multi-core and heterogeneous architectures. Several Commercial-Off-The-Shelf (COTS) solutions already exist that implement such advanced features, and space agencies have shown great interest in integrating them [18, 20]. However, COTS components do not often have the necessary safety properties, in terms of reliability and timing. Thus, their use in critical scenarios presents numerous challenges, especially for processors. Indeed, due to safety requirements, COTS platforms must ensure system correctness, which can be split into logical correctness and temporal correctness. The former requires the output to be correctly computed, while the latter requires the output to be produced within given time constraints.

CF '23, May 9–11, 2023, Bologna, Italy

ACM ISBN 979-8-4007-0140-5/23/05.
https://doi.org/10.1145/3587135.3592765

1.1 Logical correctness

Guaranteeing logical correctness starts with the software development and testing phases. These human-based processes are prone to errors, which may lead to software bugs and hardware design defects. In this article, we, instead, focus on natural random faults – particularly, Single Event Upsets (SEUs) – and we voluntarily neglect the discussions related to human-caused faults, which are not the subject of this work. Random hardware faults are unpredictable and their sources are difficult to manage. Indeed, they are caused by three main sources: cosmic rays, chip package impurities, and other radiation sources exist. Cosmic rays generate heavy-ion particles that hit the memory cell, material impurities in the chip package may be composed of radioactive elements which decay and produce radiation, and other nuclear radiation sources. These events cause SEUs, which are usually modeled as a bit-flip in a memory component. In space, due to the lack of Earth's atmosphere and magnetic field, these problems are exacerbated. On unprotected devices, several SEUs are experienced during one day [5, 28].

It is possible to reduce the frequency of these faults, but not to eliminate them. Indeed, a mechanical shield against cosmic rays can be implemented but the reduction has a positive correlation with the weight of the shield, in contrast with the goal of reducing weight in aerospace applications. The amount of package impurities can be reduced by improving manufacturing processes, but it is still impractical to eliminate all of them. Other radiation sources are often difficult to control and have the same shielding problem. Consequently, we can take (costly) countermeasures to reduce the source of the faults but we still need to consider the probability of them occurring and take actions to reduce the safety risk.

To increase the reliability of safety-critical systems to such random faults, hardware solutions are traditionally employed. Redundancy, at different levels, is the most common way to achieve resilience. The use of specialized hardware techniques is, however, in contrast to the goal of introducing COTS in critical systems. On the other hand, COTS components are general-purpose hardware not usually designed to include fault tolerance elements [24].

1.2 Temporal correctness

Almost all safety-critical embedded systems are also *real-time systems*. Indeed, the output must be correct and also produced within given time constraints. For instance, a fly-by-wire controller must

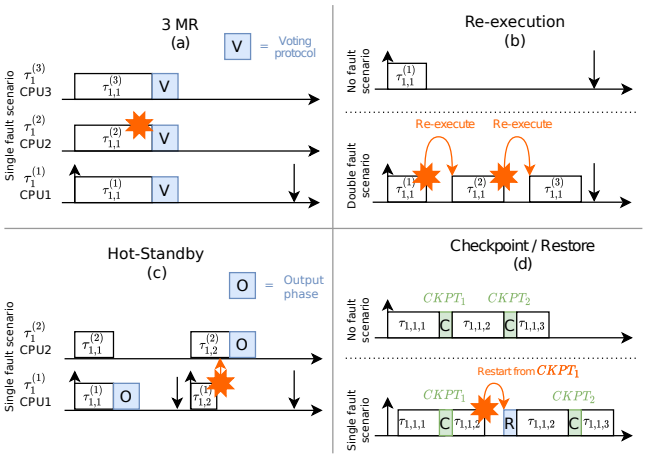

Figure 1: Example of four SIHFT techniques.

obviously produce correct output, but it should do that also in a timed way, otherwise, the flight safety can be compromised. Most of the certification processes require to formally prove the satisfaction of these constraints under any operational conditions. Such a proof requires to perform a scheduling analysis to verify that all software tasks are able to finish by their deadlines. Performing a correct scheduling analysis requires knowledge of the hardware architecture timing characteristics and a model of the software behavior. The latter is for sure a challenging analysis, but many techniques have been developed in the last decades. This is not the case of the hardware model: the more the complexity of the hardware architecture, the more difficult is to obtain a realistic estimation of the hardware timing characteristics and apply them to the software. In particular, the *Worst-Case Execution Time (WCET)* is the software metric that represents the maximum execution time that a software task needs to complete in the worst condition. Its estimation depends on the software itself but also on the hardware timing. Obtaining the exact value of the WCET with static (analytical) methods is not usually feasible in modern computational platforms, due to the high computational complexity of the analysis. Let us call $\overline{C}_i$ the estimation of the real WCET C_i of a task τ_i. Then, the estimation is said to be *safe* if $\overline{C}_i \geq C$. The estimation is said to be *tight* if $\overline{C}_i - C$ is *not too large*. The term is voluntarily vague because it depends on the specific scenario. Having a non-tight WCET is not a safety issue but it makes the system inefficient, wasting computing resources that cannot be allocated to other tasks.

2 ENABLING TECHNOLOGIES

To introduce COTS systems in safety-critical applications, it is necessary to guarantee logical and temporal correctness, which is challenging due to the nature of the COTS itself. The next sections provide an overview of three technologies that can enable the use of COTS for safety-critical systems, particularly space systems.

2.1 SIHFT

Approaches that implement fault tolerance to hardware faults in software are called *Software-Implemented Hardware Fault Tolerance (SIHFT)* and include:

- *Fault detection* techniques, such as range checks of the input/output values, plausibility checks of the output values, error detection codes, an external monitoring device, control flow monitoring, and watchdogs. The goal of the detection mechanisms is to identify, as soon as possible, the occurrence of a fault by looking at the temporal and logical properties of the running software tasks. In some cases, fault detection can also be provided by the hardware. For instance, when a SEU in a pointer variable causes a memory violation access.
- *Fault recovery* techniques, which are categorized in *space redundancy* and *time redundancy*. The former includes task replicas and standby tasks strategies: each task runs replicated multiple times (possibly on different processing units); then, a voting procedure decides which is the correct result among the task replicas. Fault detection techniques can also be used to determine which one is the correct output. The standby strategy consists in performing the full execution of the replicated tasks only if it is necessary. The *time redundancy* category, instead, includes approaches that react to a fault detected by the fault detection. In this case, the most common approaches are: re-execution (sometimes called *retry mechanism*), checkpoint/restore (sometimes called checkpoint/restart), recovery blocks, code correcting codes, and forward and backward error recovery.

Figure 1 shows four examples of fault recovery approaches implemented as SIHFT: (a) a triple modular redundancy scenario, where the three replica tasks run on different processors and a voting procedure is performed; (c) a stand-by strategy, where the replica task performs the computation but not the output phase, unless a fault occurs; (b) a time redundancy mechanism, where tasks are re-executed if a fault is detected; and (d) the checkpoint/restore.

Impact. SIHFT approaches are attractive to implement fault tolerance against hardware faults. Indeed, the idea of SIHFT is to avoid the use of specialized hardware designs to implement fault tolerance. Moving the hardware fault tolerance to the software layer has many advantages: the reduction of the design and implementation costs of the hardware and more flexibility in the development process, because any modification requires a software (and not hardware) change. Consequently, SIHFT enables the use of COTS computing platforms in critical scenarios. The main disadvantage of SIHFT is the increased computational workload. This overhead often consists of more than 100% of the original workload utilization, for each replica or re-executed portion. This disadvantage is also shared by hardware fault tolerance, which may have a > 100% overhead in terms of power, performance, and area [6]. Therefore, SIHFT remains very attractive even considering this overhead.

2.2 Mixed-Criticality Systems

Mixed-Criticality (MC) is a concept born in academia at the beginning of the 2000s, that has been interpreted with different meanings throughout the years. Vestal's paper in 2003 [29] proposed the most common MC model. The idea is the following: because the static estimation of a safe WCET value C_i usually leads to over-pessimistic estimations, we estimate several values for the WCET – i.e., $C_i^1, C_i^2, ..., C_i^{\chi_i}$ – at different levels of assurances depending on the *criticality* level of the task χ_i. For instance, in a two-criticality

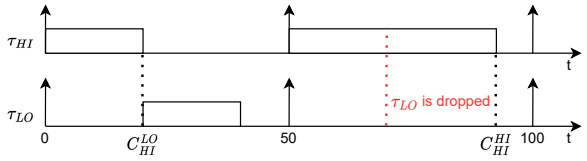

Figure 2: An example of execution of a MC workload composed of two tasks.

level system, we have HI and LO-criticality tasks. Let us assume that the system is composed of only two periodic tasks, τ_{HI} and τ_{LO}, having period 50, LO-criticality WCET $C_{LO}^{LO} = C_{HI}^{LO} = 20$, and HI-criticality WCET $C_{HI}^{HI} = 40$. Two cases, depicted in Figure 2, may happen at run-time: τ_{HI} completes before C_{HI}^{LO} (the first period in the figure), or τ_{HI} completes after C_{HI}^{LO} and before C_{HI}^{HI} (the second period in the figure). In the former case, we can admit also τ_{LO} in the schedule, while in the latter we have to drop τ_{LO} to allow τ_{HI} to run. This example is only a trivial case: several scientific works were inspired by Vestal's article and they are surveyed in a constantly-updated survey by Burns et al. [3, 4].

While these approaches showed effectiveness in improving the system utilization and reducing the resource waste caused by the WCET overestimation, they have never been widely used in industry. Besides the applicability problems in common to many academic real-time scheduling algorithms, the MC concept used in academia diverged compared to the MC used in industry and approved by standards [8]. The main objection is related to the dependence created between the tasks of different criticality. Indeed, in the example of Figure 2, a task of HI-criticality WCET caused the failure of a task of LO-criticality. This sort of dependence is not usually allowed by standards.

Impact. Consolidating the computing units is a key goal to reduce the cost of avionics in aerospace applications. The cost has to be intended not only as the monetary cost of the system itself, but also as a series of non-functional metrics that impact the feasibility of the whole system. In many scenarios, the actual system utilization of computing devices is very low in practice and for most of the operational time, because the WCET of tasks is often overestimated and almost never occurs in practice. This has the effect of observing the system to be idle for most of the time, wasting computing resources and related metrics (power consumption, number of systems needed, etc.). MC can partially solve this issue, thanks to the multiple estimations of the WCET values.

2.3 Probabilistic timing analysis

The necessity to reduce the WCET pessimism led the researcher to study, since 2001 [7, 2], the possibility to estimate the WCET with probabilistic methods. In particular, the measurement-based methods attracted significant interest in both academia and industry thanks to the easiness of the implemented approach. They are usually based on the statistical theory named Extreme Value Theory (EVT). The estimation process is the following: a large set of time measurements of the task is collected and a statistical estimator provides the probability distribution of the following expression: $\bar{p} = P(\mathcal{E} > C)$ where p is the violation probability, C is a WCET threshold and $\mathcal{E}$ is the random variable representing the execution

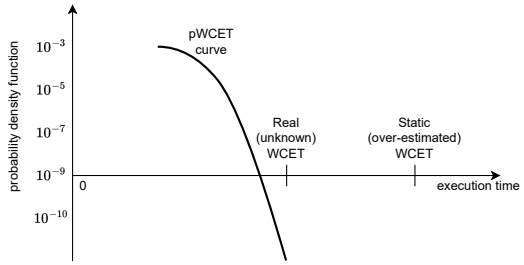

Figure 3: An example of a pWCET distribution curve compared to real and statically-estimated WCETs.

time. The key concept of EVT is that the tail of the distribution of an observed phenomenon is always distributed according to a well-known distribution, named Generalized-Extreme Value (GEV) distribution, independently from the original distribution of the input samples. This mathematical result can be mapped to the concept of execution time: independently from the execution time distribution, the WCET is always distributed according to the GEV. Hence, the use of EVT allows the estimation of very small values for p (e.g., 10^{-9} or even lower) without the need to observe billions of samples to build the cumulative distribution function.

The output of this WCET estimation process is therefore the pWCET, which is no more a scalar value like the WCET, but a statistical distribution from which it is possible to derive the WCET C given a required violation probability p or, vice versa, the violation probability p given a desired WCET C. An example of the probability density function of this distribution is depicted in Figure 3.

Impact. The availability of a measurement-based method to estimate the WCET enables the use of COTS-based systems. Indeed, manufacturers of COTS-based systems usually do not provide accurate timing models for the WCET of the hardware architecture. For example, many DRAM memory controllers do not have an upper bound for the latency in accessing a memory cell, but only an average value. In such a case, static WCET timing analyses are not usable. Additionally, the probabilistic method reduces the pessimism in the WCET estimations introduced by the simplification needed by the current hardware abstraction. This, in turn, enables the exploitation of the computational power of complex computing platforms, also for safety-critical workload.

3 ANALYSIS OF THE CURRENT STANDARDS

Introducing SIHFT, mixed-criticality, and probabilistic real-time in critical systems is a challenging activity that needs modifications of the standards, which requires, in turn, changes in established best practices. This process is not only long but also difficult, and, sometimes, academic and industry concepts diverge so much that years of academic products are inapplicable in practice. Mixed-criticality is, among the three technologies, the most iconic example: in academia, the concept of criticality has taken a different path for a long time with respect to the criticality used in safety standards. This distinction has been highlighted by Ernst et al. in 2016 [8] and many papers published nowadays are still based on the *academic version* of criticality. We will detail this issue later in Section 4.

In this section, we survey relevant standards and handbooks used by the European Space Agency (ESA), understanding which

Table 1: Relevant standards analyzed in this article.

ID	Title	Ref.
ECSS-E-ST-40C	Space Engineering Software	[11]
ECSS-E-HB-40A	Space engineering - Software Engineering Handbook	[13]
ECSS-Q-HB-80-03A	Space product assurance - Software dependability and safety	[12]
ECSS-Q-ST-60-13C	Space product assurance - Commercial electrical, electronic and electromechanical (EEE) components	[9]
ECSS-Q-HB-60-02A	Space product assurance - Techniques for radiation effects mitigation in ASICs and FPGAs handbook	[15]
ECSS-Q-ST-80C	Software product assurance	[14]
DO-178B/C	Software Considerations in Airborne Systems and Equipment Certification	[27]

are the imposed limits and the best practices related to the three technologies. The standards are part of the European Cooperation for Space Standardization (ECSS) and are summarized in Table 1. Additionally, we analyzed the aviation DO-178B/C standard.

3.1 ECSS-E-ST-40C

The ECSS-E-ST-40C standard is the main standard for space software engineering. The following sections of the standard are directly relevant:

- [11, §5.5.2.5] Detailed design of real–time software
- [11, §5.8.3.5] Verification of code
- [11, §5.8.3.11] Schedulability analysis for real-time software
- [11, §Annex F] Software Design Document (particularly the real-time model)
- [11, §Annex R] Tailoring of this Standard based on software criticality

We do not provide further details in this section, because the interpretation of the exact sentences of the standard is integrated into the following descriptions of the handbooks.

3.2 ECSS-E-HB-40A

The ECSS-E-HB-40A handbook [13] provides interpretations and best practices for the implementation of the requirements specified in ECSS-E-ST-40C. The relevant parts of this handbook for this paper's discussion are summarized in the next paragraphs.

3.2.1 Criticality definition. In the flight software characterization section [13, §4.2.3], the term *criticality* is defined as *"The criticality of a software product is determined at system level, based on the severity of the consequences of the system failures that the software can cause"*. Moreover, it is specified that *"no failure propagation is possible from lower-criticality components to higher-criticality components"*. This requirement matches the real-time criticality uses, where the overrun of lower-criticality tasks cannot make the higher-criticality task to miss their deadline. The handbook defines 4 levels of criticality: A, B, C, and D (the latter refers to non-critical software).

3.2.2 Design of real-time software. Section [13, §5.5.2.5] describes which aspects compose the timing properties of the software system (scheduling algorithm, inter-task communication), their constraints (e.g., max jitter, max CPU usage, max response time), and the dynamically allocated resources. Then, the handbook states that the *"analysis of these real-time elements should permit to determine whether the timing constraints on the system can be satisfied."*.

3.2.3 Software testing - Timing properties and fault injection. Section [13, §6.4.4.3] describes *Performance testing* (which also includes CPU usage and timing). Section [13, §6.4.6] discusses how to verify that real-time software is able to meet the performance criteria. Three possible methods are described: logging (the software itself logs any time constraints violation), monitoring outputs (with external components such as logic analyzers), and emulation. The handbook recommends using a combination of these three methods. It also recognizes the difficulties in finding realistic worst-case scenarios and the WCET: *"Real time testing cannot be exhaustive. It needs to be complemented with schedulability analysis based on WCET. Real time software testing is necessary to increase the confidence and provide real individual loads figures (to feed the real time analysis). It should provide the evidence of comfortable margins w.r.t. schedulability analysis which is too much theoretical/pessimistic."*

3.2.4 Schedulability and WCET. Two different WCET definitions are provided in section [13, §7.2.2.2]: *theoretical WCET* (measured, estimated, or derived from the code analysis technique) and *operational WCET* (refinement of the theoretical one – i.e., smaller – by considering only realistic operational scenarios, such as nominal, degraded, per-mission, per-phase, etc.).

3.2.5 Computational model for real-time software. In section [13, §7.4.3], the handbook defines (and describes) the following single-core computational models for real-time tasks: cyclic executive, preemptive, preemptive without cyclic preemption, Ravenscar Computational Model (RCM), and partitioned systems. Cyclic executive and preemptive scheduling algorithms are the usual well-known techniques matching the academic literature. The *preemptive without cyclic preemption* algorithm forces all the tasks to have the same period in order to avoid the preemption due to task activation, with the exception of asynchronous event routines (e.g., interrupt handlers). RCM is a scheduling model which is linked with the Ravenscar ADA profile, thus specific to the ADA programming language. It is similar to preemptive techniques but with tighter constraints. Finally, the *partitioning* strategy allows the logical separation of software building blocks with different integrity levels. The handbook states that there are "on-going investigations" on the execution environment that provides the partition strategy. Further investigations on the current status and on the relations/differences with mixed-criticality are needed.

3.3 ECSS-Q-HB-80-03A

The ECSS-Q-HB-80-03A handbook is the document of major interest for fault-tolerance aspects. Section [12, §4] defines the terminology (with the usual fault/error/failure nomenclatures) and section [12, §5] the general workflow and safety framework. All the processes are clearly distinct between two levels: software-level and system-level.

Section [12, §5.2.3.2] describes how software criticality is defined at system-level and the cases when it is possible to reduce the software criticality, i.e., when compensating provisions are available (e.g., electromechanical backups, software monitors, etc.). It also discusses the criticality inheritance mechanism: the criticality category to be assigned is directly linked to the function criticality and if a compensating provision mechanism is implemented, then it inherits the original criticality of the component.

Section [12, §5.2.3.3] defines the software-level criticality definition. It discusses how software-level choices may impact on criticality. For instance, the criticality of two software components sharing the same memory address space is the highest criticality among the twos. The results of the software-level analyses are documented in a software dependability and safety analysis report, which is fed back to the system-level analysis in order to: 1) identify software failures that might have impact on the system-level criticality, 2) provide specific recommendation for the system-level activities to drive a change in the system architecture model.

Complementary to these activities, the *Hardware-Software Interaction Analysis (HSIA)* (section [12, §5.2.5]) verifies how the software reacts after hardware failures. This analysis is performed at system-level but requires a significant support from software developer. For each potential hardware failure that might affect software, it is mandatory to provide a a set of requirements that define the software behaviour in the occurrence of that hardware failure.

It is worth mentioning that among the techniques to increase fault tolerance, the N-version programming to solve the common cause failures is *not* a recommended method [12, §6.4].

3.4 ECSS-Q-ST-60-13C

The ECSS-Q-ST-60-13C standard [9] (in conjunction with ECSS-Q-ST-60C [10]) is related to COTS components, and the focus in entirely on hardware characteristics and testing (including procurement, temperature stress tests, mechanical analysis, etc.).

3.5 ECSS-Q-HB-60-02A

The handbook ECSS-Q-HB-60-02A [15] summarizes the techniques used to mitigate the effect of radiation-induced faults. The document mostly focuses on hardware fault tolerance techniques, with the exception of the section on SIHFT [15, §14]. The handbook emphasizes the importance of SIHFT in the context of COTS hardware to reduce the costs but maintain reliability to SEE. Software techniques are categorized in: 1) redundancy at instruction level, 2) redundancy at task level, and 3) redundancy at application level. All these techniques are based on the instruction/task/application replica+voting schema. The handbook describes in detail such techniques. It is also worth mentioning the following sections [15, §15], explaining system-level protections, such as watchdogs, error correcting codes, and other techniques, and section [15, §16], dealing with validation methods and, in particular, fault injection.

3.6 ECSS-Q-ST-80C

The quality standard ECSS-Q-ST-80C [14] describes the requirements to be used for the development and maintenance of space software. Paragraph [14, §6.2.2.9] describes the necessity to verify that the software correctly reacts to hardware failures without leading to system failures, as specified in the HSIA document. The subsequent paragraph [14, §6.2.2.10] focuses on criticality concept, as we already discussed in Section 3.3. The standard also specifies how criticality is defined: Table D-1 in [14, §Annex D] provides the definitions of each criticality level and how to map it to the "category functions" classification.

3.7 DO-178B/C - Aviation[1]

The DO-178B [27] and DO-178C present more general considerations on software characteristics then the other standards. The timing properties of the software are cited in objective [27, §6.3.4], which requires the reviews and analyses to be accurate and consistent, and in [27, §6.4.3] where the requirements-based hardware/software integration testing can be used to find violations of the worst-case execution time estimations. In the DO-178C a clarification is added to specify that also compiler and linker, and their options, and hardware features affecting the execution time must be considered in the WCET analysis. The DO-178C also introduces measurement-based analyses but still requires to have a rigorous demonstration that the timing measurements triggered the worst-case condition [21]. The white paper by Rapita Systems describes the challenges in multi-core timing analysis under DO-178C [22].

Paragraph [27, §2.2.2] states that software levels are not related to the software failure rate, i.e., the software reliability rates cannot be used by system safety assessment process as can hardware failure rates. Indeed, the the failure probabilities are assigned by the DO-254 standard to the hardware. It also states that N-version programming can be used as extra safety measure, but its improvement cannot be quantified in terms of failure probabilities ([27, §2.3.2]). Finally, software partitioning is also mentioned and each partition has the same criticality of the highest of its members. There is no mentioning about dependency between partitions (they are assumed independent).

4 THE IMPACT ON SAFETY

4.1 Are the proposed technologies compliant with the standards?

4.1.1 SIHFT. Most of the standards allow the use of SIHFT to reach the reliability goal, but only in a qualitative way: they do not allow the software to have a defined probability of failure, preventing to use SIHFT for the demonstration of the quantitative goal of reliability against hardware faults. For instance, DO-178B/C explicitly specifies that software levels are not assigned to specific failure rate probability like the hardware failure rates. This non-quantifiability is due to the fact that software failures are usually considered software bugs. Indeed, it is difficult (if not impossible) to estimate the probability that a bug exists. Vice versa, in SIHFT, the probabilities come from the hardware fault probabilities, therefore they are quantifiable, provided that a careful analysis is performed. The use of such probabilities can be a first step in the reconciliation process with hardware and software failure probabilities.

4.1.2 Mixed-Criticality. The definition of criticality and its classification is similar among all the standards. ECSS has 4 criticality levels (A-D, A is the highest) and DO-178B/C has 5 criticality levels

[1]Note: Paragraph reference numbers in this section refer to DO-178B, not DO-178C.

(A-E, A is the highest). Similarly, in automotive, ISO 26262 [19] specifies 4 criticality levels (A-D, D is the highest) and, in railways, EN 50128 [16] specifies 5 criticality levels (0-4, 4 is the highest).

All the standards agree on the *criticality propagation* concept: when a failure dependency exists, any software component that may cause the failure of a software component of criticality X must be at least of X criticality. This is expected and matches most all scientific works. However, the standards do not explicitly consider the opposite case: what happens if a task of higher criticality negatively impacts the execution of a lower criticality task? This is a key problem that exists when the academic approach to mixed-criticality is used in the industrial context. Standards specify how to manage co-running tasks on the same device, especially when tasks with different criticalities co-exist on the same platform. ECSS-E-HB-40A suggests the *partitioning* strategy to allow the logical separation of software building blocks with different integrity levels.

The answer to the question of whether traditional MC is applicable or not relies on if higher criticality can affect the execution of lower criticality or not. MC differs from *partitioning* because tasks share the same resources (including time) and therefore may interfere with each other. The standards generally describe "software-dependent failures": we can consider in this category a higher-criticality task overrunning its low-criticality WCET causing a failure propagation to another task of lower-criticality. The problem is how to evaluate the probability that this event occurs: the probability that a task overruns its low-criticality WCET is hard to estimate. Therefore, it seems difficult to determine the plausibility of this event (as required by, for instance, ISO 26262).

4.1.3 Probabilistic real-time. The specification on how the WCET estimation should be performed is almost identical in all the analyzed documents. In fact, ECSS-E-HB-40A requires to demonstrate by analysis whether timing constraints can be satisfied. The same requirement is present in DO-178B/C. All these standard allow the use of measurement-based techniques as *testing* to find possible violation (as an extra safety measure), but a static analysis is still required as proof of correctness (or the measurement-based must be formally provided proof of correctness). No safety-critical standards mention probabilistic analyses.

4.2 Possible standard changes

4.2.1 SIHFT. Linking the probability of a fault occurring in the hardware to the software failure is mathematically possible. For instance, if a transient fault occurs in hardware with a given probability, this fault event can be extended to a fault event in the software. Then, if software fault tolerance techniques are in place, the failure probability of the software can be reduced. Recent works studied how to compute this exact software failure rate caused by hardware [25, 23]. For this reason, we believe that standards should evolve in allowing the use of SIHFT in a quantitative way with respect to the goal of reliability. Clearly, a SIHFT technique should be approved for quantitative use, only if an analytical proof of the quantitative improvement is provided.

4.2.2 Mixed-Criticality. Baruah [1] believes that the problem of non-applicability of MC to real systems is actually a matter of changing the current practices (and standards), rather than real limitations of the MC theory. Indeed, allowing higher-criticality tasks to cause a graceful degradation of lower-criticality tasks looks a reasonable mechanism. In some cases, this dependence can be analytically analyzed, providing a quantitative bound of the impact on lower-criticality tasks [25]. While not immediately applicable like SIHFT, we believe that a discussion on how to introduce MC in the standards should start, especially in spacecraft systems where the consolidation of multiple platforms is a key optimization goal.

4.2.3 Probabilistic real-time. In our opinion, the full transition to measurement-based approaches to estimate the WCET is not a valid choice for safety-critical software (yet). While static analyses cannot cope with the increasing complexity of the hardware, measurement-based approaches (both deterministic and probabilistic) cannot guarantee to have obtained sufficient information content to provide a reliable estimation. However, exploiting measurement-based approaches for small parts (such as at the instruction-level or basic block-level) or as a comparison (e.g., to verify "how far" the static WCET is far from the real one) can be advantageous. The other possibility is to introduce probabilistic estimations only for lower-critical workloads; however, it should be accepted that it is not possible to demonstrate safe quantitative numbers yet.

5 FUTURE DIRECTIONS AND CONCLUSIONS

SIHFT approaches are already used in industrial safety-critical systems, even if they are used to provide a "qualitative reliability". We believe that novel analyses and tools that increase the level of detail on how SIHFT impacts on the reliability metrics can be a good starting point for the introduction of SIHFT in standards. In parallel, the study of novel SIHFT techniques, possibly transparent to the developer, can improve such quantitative benefits.

Mixed-criticality is another technology with a good level of maturity, and many approaches are formally proven and implemented in several research prototypes. More work needs to be done on solving practical implementation issues and on the relation with standards, especially how to handle graceful degradation.

Probabilistic real-time is, for sure, the technology of the three with the lowest technology readiness level, because theoretical problems still exist [26]. However, the feeling in several members of the scientific community is that we cannot advance without including statistical property in the systems. Probabilistic or not, the WCET problem is the most crucial issue to introduce modern computing platforms in critical systems.

In conclusion, critical systems, and especially spacecraft systems, need to guarantee logical and temporal correctness. We showed three technologies that may help in reaching these two goals in modern COTS computing platforms, we discussed their advantages and limitations, and we compared them with the current certification processes, highlighting the incompatibilities and possible changes that can be introduced in standards.

ACKNOWLEDGMENTS

This work has received funding by the European Space Agency (OSIP no. 4000133770 / 21 / NL / MH / hm) and the ICSC National Research Center in High-Performance Computing.

REFERENCES

[1] Sanjoy Baruah. 2018. Mixed-criticality scheduling theory: scope, promise, and limitations. *IEEE Design & Test*, 35, 2, 31–37. DOI: 10.1109/MDAT.2017.2766571.

[2] G. Bernat, A. Colin, and S. M. Petters. 2002. WCET analysis of probabilistic hard real-time systems. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.* IEEE, (Dec. 2002), 279–288. DOI: 10.1109/REAL.2002.1181582.

[3] Alan Burns and Robert Davis. 2013. Mixed criticality systems-a review, 1–69. Constantly updated. https://www-users.york.ac.uk/~ab38/review.pdf.

[4] Alan Burns and Robert I. Davis. 2017. A survey of research into mixed criticality systems. *ACM Comput. Surv.*, 50, 6, Article 82, (Nov. 2017), 37 pages. DOI: 10.1145/3131347.

[5] A. Campbell, P. McDonald, and K. Ray. 1992. Single event upset rates in space. *IEEE Transactions on Nuclear Science*, 39, 6, 1828–1835. DOI: 10.1109/23.211373.

[6] Athanasios Chatzidimitriou, George Papadimitriou, Christos Gavanas, George Katsoridas, and Dimitris Gizopoulos. 2019. Multi-bit upsets vulnerability analysis of modern microprocessors. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*, 119–130. DOI: 10.1109/IISWC47752.2019.9042036.

[7] S. Edgar and A. Burns. 2001. Statistical analysis of WCET for scheduling. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, 215–224.

[8] Rolf Ernst and Marco Di Natale. 2016. Mixed criticality systems—a history of misconceptions? *IEEE Design & Test*, 33, 5, 65–74. DOI: 10.1109/MDAT.2016.2594790.

[9] ESA-ESTEC. 2021. Commercial electrical, electronic and electromechanical (EEE) components – Rev.1. Standard ECSS-Q-ST-60-13C. European Cooperation for Space Standardization, Noordwijk, The Netherlands, (Aug. 2021).

[10] ESA-ESTEC. 2013. Electrical, electronic and electromechanical (EEE) components – Rev.2. Standard ECSS-Q-ST-60C. European Cooperation for Space Standardization, Noordwijk, The Netherlands, (Oct. 2013).

[11] ESA-ESTEC. 2009. Software. Standard ECSS-E-ST-40C. European Cooperation for Space Standardization, Noordwijk, The Netherlands, (Mar. 2009).

[12] ESA-ESTEC. 2017. Software dependability and safety – Rev.1. Handbook ECSS-Q-HB-80-03A. European Cooperation for Space Standardization, Noordwijk, The Netherlands, (Nov. 2017).

[13] ESA-ESTEC. 2013. Software engineering handbook. Handbook ECSS-E-HB-40A. European Cooperation for Space Standardization, Noordwijk, The Netherlands, (Dec. 2013).

[14] ESA-ESTEC. 2017. Software product assurance – Rev.1. Standard ECSS-Q-ST-80C. European Cooperation for Space Standardization, Noordwijk, The Netherlands, (Feb. 2017).

[15] ESA-ESTEC. 2016. Techniques for radiation effects mitigation in ASICs and FPGAs handbook. Handbook ECSS-Q-HB-60-02A. European Cooperation for Space Standardization, Noordwijk, The Netherlands, (Sept. 2016).

[16] European Committee for Electrotechnical Standardization. 2011. Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems. Standard EN50128. CENELEC, (June 2011).

[17] Meagan Hahn and Rachel Sholder. 2020. Cost estimation of electronic parts in nasa space missions. In *2020 IEEE Aerospace Conference*, 1–8. DOI: 10.1109/AERO47225.2020.9172574.

[18] Robert F Hodson et al. 2020. Recommendations on Use of Commercial-Off-The-Shelf (COTS) Electrical, Electronic, and Electromechanical (EEE) Parts for NASA Missions. Tech. rep. 20205011579. NASA, (Dec. 2020).

[19] International Standard Organization. 2018. Road vehicles – Functional safety. Standard ISO-26262. ISO.

[20] Mikko Nikulainen and Ferdinando Tonicello. 2021. Utilization of cots in esa missions. In *NEPP Electronics Technology Workshop*. (June 2021). https://nepp.nasa.gov/workshops/etw2021/talks/17-JUN-21_Thur/1045_Nikulainen_Tonicello-Utilisation-of-COTS-in-ESA-Missions.pdf.

[21] Rapita Systems. 2021. Automating WCET analysis for DO-178B/C. White Paper. Danlaw, York, UK.

[22] Rapita Systems. 2019. Multicore Timing Analysis for DO-178C. White Paper. Danlaw, York, UK.

[23] Federico Reghenzani and William Fornaciari. 2023. Mixed-criticality with integer multiple WCETs and dropping relations: new scheduling challenges. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference* (ASPDAC '23). Association for Computing Machinery, Tokyo, Japan, 320–325. ISBN: 9781450397834. DOI: 10.1145/3566097.3567851.

[24] Federico Reghenzani, Zhishan Guo, and William Fornaciari. 2023. Software fault tolerance in real-time systems: identifying the future research questions. *ACM Comput. Surv.*, (Mar. 2023). Just Accepted. DOI: 10.1145/3589950.

[25] Federico Reghenzani, Zhishan Guo, Luca Santinelli, and William Fornaciari. 2022. A mixed-criticality approach to fault tolerance: integrating schedulability and failure requirements. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 27–39. DOI: 10.1109/RTAS54340.2022.00011.

[26] Federico Reghenzani, Luca Santinelli, and William Fornaciari. 2020. Dealing with uncertainty in pWCET estimations. *ACM Trans. Embed. Comput. Syst.*, 19, 5, Article 33, (Sept. 2020), 23 pages. DOI: 10.1145/3396234.

[27] RTCA/EUROCAE. 1992. DO-178B - Software Considerations in Airborne Systems and Equipment Certification. Standard. RTCA/EUROCAE, (Jan. 1992).

[28] Brian D. Sierawski et al. 2017. Cubesats and crowd-sourced monitoring for single event effects hardness assurance. *IEEE Transactions on Nuclear Science*, 64, 1, 293–300. DOI: 10.1109/TNS.2016.2632440.

[29] Steve Vestal. 2007. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 239–243. DOI: 10.1109/RTSS.2007.47.