# Safety Assurance Driven Problem Formulation for Mixed-Criticality Scheduling

Patrick Graydon[2] and Iain Bate[1,2]
[1]Department of Computer Science, University of York, York, UK
[2]Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
email: patrick.graydon@mdh.se and iain.bate@cs.york.ac.uk

***Abstract*—In 2007, Vestal proposed Mixed-Criticality Scheduling (MCS) to increase utilisation despite imperfect timing evidence. Others have since refined the MCS problem formulation, proposed alternative scheduling approaches, and evaluated their performance. We assess existing MCS problem formulations from a safety assurance perspective and report problems found. Among these is the use of the word 'criticality' to mean several related but distinctly different things such as Safety Integrity Levels (SIL), importance, and confidence. We conclude with suggestions for addressing the problems found.**

## I. Introduction

In 2007, Vestal proposed Mixed Criticality Scheduling (MCS) as a way to achieve high utilisation despite imperfect timing evidence [1]. He observes that (a) increased confidence in WCET limits comes at the expense of increased pessimism and (b) tasks vary in criticality. MCS schedules a system so that all tasks can run if they adhere to a low-confidence WCET limit. Failing that, the most critical tasks can run if they adhere to a larger, higher-confidence WCET limit. Later work expanded the problem definition, proposed alternative scheduling algorithms, and demonstrated superior utilisation [2]–[10].

But existing definitions of the MCS problem are presented primarily from a scheduling perspective. We report an analysis of the MCS problem from a safety assurance perspective. Amongst other findings, we show that current formulations use the word 'criticality' to represent several related but distinct concepts, including Safety Integrity Level (SIL), importance, and confidence. We discuss how this affects safety assurance and present suggestions for addressing the problems found.

In section II, we survey formulations of the MCS problem. We discuss confidence in execution time assessments in section III and the MCS-related demands of safety assurance in section IV. In section V, we critically examine existing problem formulations and suggest improvements. We discuss future work in section VI and conclude in section VII.

## II. Existing Problem Formulations

Vestal was the first to publish a formulation of the MCS problem [1]. Others, including Baruah and Yi, have extended the problem formulation to account for more parameters changing with criticality and to generalise criticality modes.

### A. The Vestal Formulation

Vestal's 2007 formulation of the MCS problem begins with the 'conjecture that the higher the degree of assurance required that actual task execution times will never exceed the WCET parameters used for analysis, the larger and more conservative the latter values will become in practice' [1]. In his model, a system has tasks $\tau_1 \ldots \tau_n$, each with period $T_i$ and deadline $D_i$. System development adheres to 'an ordered set of design assurance levels' $\mathcal{L} = \{A, B, C, D\}$ with $A$ being the highest. $C_{i,l}$ gives the compute time for task $\tau_i$ at assurance level $l$, with $C_{i,A} \geq C_{i,B} \geq C_{i,C} \geq C_{i,D}$ for all $i$. The goal is 'to assure to level $L_i$' that each task $\tau_i$ 'never misses a deadline'.

### B. The Baruah and Burns Formulation

Baruah and Burns extend Vestal's formulation [2], [5]. A system is 'defined as a finite set of components $\mathcal{K}$', each with level of criticality $L$ [5]. As in Vestal's formulation, each task $\tau_i$ is defined by $(T_i, D_i, \vec{C_i}, L_i)$. $\vec{C_i}$ 'will be derived by a process dictated by the criticality level' and 'the higher the criticality level, the more conservative the verification process and hence the greater will be the value of $C_i$'. However, noting that 'the higher the criticality level, the greater the need for the task to complete well before any safety-critical timing constraint', the model permits different deadlines at different criticality levels so long as $L_i^1 > L_i^2 \Rightarrow D_i^1 \leq D_i^2$. Moreover, noting that 'the higher the criticality level, the tighter the level of control that may be needed', Baruah and Burns allow different periods at different criticality levels so long as $L_i^1 > L_i^2 \Rightarrow T_i^1 \leq T_i^2$. If a task at criticality level $\ell$ overruns $C_{i,\ell}$, tasks at criticality levels $\ell$ and lower are prevented from running again until the processor is idle [2].

Baruah and Burns identify two 'issues' that MCS must address: *static verification* and *run-time robustness* [5]. By the former, they mean that 'for each criticality level $\ell$, all jobs of all tasks with criticality $\geq \ell$ will complete by their deadlines in any criticality-$\ell$ behavior'. By the latter, they mean that after a transient overload, 'a robust scheduling algorithm would, informally speaking, be able to "recover" ... and go back to meeting the deadlines of lower-criticality jobs as well'.

### C. The Ekberg and Yi Formulation

Ekberg and Yi extend Vestal's formulation further [7]. Previous formulations defined criticality modes purely as a mechanism for preserving timing guarantees. Noting that 'it should be up to the system designer to decide what it means for the system to be in any one particular criticality mode', they propose using criticality modes to reconfigure systems in

response to events such as hardware failures. (General models for such reconfiguration have been proposed elsewhere [11]; the contribution of Ekberg and Yi lies in combining MCS with more general notions of reconfiguration.) In their model, a system is defined by a set of tasks $\tau$ and a DAG $G$. The vertex set $V(G)$ defines the criticality modes and the edge set $E(G)$ specifies the permissible changes between them. Each task is defined by the set of criticality modes in which it is active, $\mathfrak{L}$, and for each $m \in \mathfrak{L}_i$ a tuple $(C_i(m), D_i(m), T_i(m))$. During a criticality level switch from $m$ to $m'$:

- Where $m \in \mathfrak{L}_i$ and $m' \notin \mathfrak{L}_i$, the system suspends $\tau_i$.
- Where $m \notin \mathfrak{L}_i$ and $m' \in \mathfrak{L}_i$, the system activates $\tau_i$.
- Where $m \in \mathfrak{L}_i$ and $m' \in \mathfrak{L}_i$, the system changes $\tau_i$'s parameters to those specified for $m'$.

These changes take immediate effect. For example, if the system suspends $\tau_i$, it discards any active jobs.

Note that $G$ is a DAG: if the system switches to a higher-criticality mode, it will never switch back. Ekberg and Yi observe in a footnote that 'one could easily find a time point where it is safe to switch back', but leave this to future work.

## III. Confidence in Execution Time Assessments

A key assumption behind all three MCS problem formulations is *WCET confidence monotonicity*: that the degree to which a WCET limit overestimates true WCET increases monotonically with confidence. WCET assessment approaches can be classified as (a) *dynamic* (measurement-based), (b) *static* (analysis-based), or (c) *hybrid* [12]. We consider the WCET confidence monotonicity assumption with respect to each WCET assessment approach in turn.

### A. Dynamic Approaches to WCET Assessment

The simplest dynamic WCET assessment approach is *High Water Mark testing* (HWM). In HWM, analysts use the longest execution time observed in testing as a WCET estimate [12]. The primary source of uncertainty in WCET limits derived from HWM testing is imperfect test coverage. Secondary sources include the correctness of any tools, the integrity of data gathering, any differences between the test system and the deployed system. These sources are epistemic, i.e. related to what we do not know rather than arising from chance.

Developers sometimes analyse execution paths and choose test cases in an attempt to stimulate the worst case [1]. If $a$ and $b$ are sets of test cases and $a \subset b$, then HWM testing using $b$ inspires more confidence than the same testing using $a$. Unfortunately, it is not generally possible to quantify either the likelihood or degree of underestimate using this technique [12]. Two examples of why are as follows. Firstly, the tasks are analysed for their WCET in isolation with the cache flushed, however the cache is not flushed before the task starts executing. Cache-related anomalies exist which mean that this situation can lead to a higher execution than the measured WCET [13]. Secondly, the WCET is measured during system integration testing however this cannot for instance cover all initial cache states and preemptions scenarios.

In probabilistic approaches, analysts use Extremal Value Theory (EVT) to fit observed execution times to a distribution [12], [14]. By selecting a WCET estimate from further to the right of the distribution, a developer can reduce the uncertainty *from test coverage* at the cost of increased pessimism. Because the distributions do not model epistemic uncertainty, probability figures taken from them are not complete descriptions of the confidence testing should inspire. However, we know of no reason to think that total confidence does not rise monotonically with distribution-derived probabilities.

### B. Static Approaches to WCET Assessment

Static WCET analysis considers all possible paths through analysed code. Thus, it is not affected by the main source of uncertainty in dynamic approaches. However, static analysis does not produce *perfect* confidence [12]. User inputs such as loop bound limits might be wrong, tools might be buggy, and processor models may be wrong.

Because primary sources of uncertainty in static approaches are epistemic, there is no clear relationship between confidence and overestimate amongst static approaches. Consider two static analysis tools $A_1$ and $A_2$ and a program $p$ for which $A_1$ produces a greater WCET figure than $A_2$. The extra overestimate might mask some tool defects (if any), but if the tool qualification evidence for $A_2$ is more comprehensive than that for $A_1$, we might justifiably have more confidence that $A_2$ does not underestimate true WCET.

We can't quantify test coverage uncertainty in HWM-based approaches. But all other issues (e.g. developer competence, tool qualification evidence) being equal, static approaches produce greater confidence. Since a static analysis approach (if it is sound, given correct inputs, etc.) never underestimates WCET and HWM testing never overestimates, static approaches will overestimate more. The degree of overestimate will depend on the target CPU's complexity and the analysis tool's sophistication. Analysis results within 20% of the highest observed execution time have been achieved [15], but complex features such as multi-level caches can result in overestimates of 100% or more [16]. Analysis of software on multicore platforms is particularly challenging [12].

It is not clear how confidence and overestimation compare between static approaches and probabilistic approaches, at least at the high end of the distribution.

### C. Hybrid Approaches to Determining WCET

Hybrid approaches combine static analysis and measurement [12], [17]. These approaches divide the software into blocks, dynamically measure the runtime of each, statically analyse the software structure, then combines block timing figures into a complete WCET estimate. The simplest form of hybrid approach uses a HWM time for each block, but more complex analyses take inter-block dependencies into account [18]. For a given test suite, hybrid approaches should be less likely to underestimate WCET than simple HWM testing: many test cases might provoke a given block's worst performance, but only one case yields the worst performance for the

task being tested. However, the basic sources of uncertainty are the same as with dynamic methods. Underestimation and overestimation are possible and their likelihood unknown [12]. It is less clear how either the confidence inspired by hybrid approaches or the degree of overestimate compares to those of static approaches. In addition, no proof of safeness exists for the hybrid approaches.

### D. The WCET Confidence Monotonicity Assumption

Between the HWM testing approach and any sound, well-constructed static analysis approach, the WCET confidence monotonicity assumption almost certainly holds. With other approaches, it is less clear that this assumption holds. We return to this issue and its implications in subsection V-A.

## IV. The Demands of Safety Assurance

Successful instantiations of MCS in safety critical systems must provide the properties and evidence needed for safety and safety assurance. In this section, we outline the timing-related demands of typical safety assurance approaches.

Standards for software for use in safety critical systems vary. For example, software for use on commercial air transports is constructed in conformance to RTCA DO-178B [19] or its successor DO-178C [20]. Software for general safety-related systems is often developed in conformance to IEC 61508 [21]. Software for road vehicles is developed in conformance with ISO 26262 [22]. There are standards for rail applications [23] and other specific domains. In yet other domains, standards dictate a systems safety approach but no specific approach to software [24]. In this section, we consider *typical* safety assurance approaches, referring to common standards for concrete examples where appropriate.

### A. Derivation of Timing-Related Safety Requirements

Engineers typically derive timing-related safety requirements from hazard and risk analyses. For example, engineers following IEC 61508 (1) define the system concept and scope, (2) perform hazard and risk analysis, (3) define overall safety requirements, and (4) allocate safety requirements to particular functions and subsystems[1] [21]. The aim of hazard and risk analysis is to determine (a) the system hazards, (b) the event sequences that could lead to those, and (c) the risks related to the identified hazards. The overall safety requirements detail risk reduction measures meant to achieve functional safety targets. Engineers allocating safety requirements specify an appropriate SIL[2] for each function and then for the subsystems implementing those functions. Timing-related safety requirements capture when these functions must be activated, performed, completed, etc. if the system is to meet its safety targets. Most standards for software in safety critical systems use some variant of this process. (RTCA DO-178B [19] and DO-178C [20] do not: they exclude system safety engineering from their scope. These standards are used with SAE ARP 4754A [25], which does follow a broadly similar pattern.)

[1]IEC 61508 uses the term 'system' here, but this process applies to multiple interacting 'systems' and considers over-arching safety goals.

[2]SILs go by different names (e.g. ASIL, DAL, SWDAL) in other standards.

### B. The Meaning and Role of Safety Integrity Levels

SILs play a complex and frequently misunderstood role in safety standards that use them [26]. SIL is *not* a measure of target reliability generally: if failure of a subsystem would have little effect on safety but a disastrous effect on business objectives, that subsystem might have a low SIL. SIL is *not* a measure of the importance of any particular property: a high-SIL function might cause harm if not performed correctly but not if completed late. SIL is also *not* a measure of *achieved* reliability. In IEC 61508 [21] (and also in ISO 26262 [22]), SIL drives which techniques (e.g. use of formal methods or dynamic reconfiguration) the standard *recommends*. However, developers are not *obligated* to use even highly recommended techniques: they may instead explain their reasoning to an assessor and agree an alternative process [21]. Even if software development process was a strong predictor of reliability – there is little or no direct evidence showing that it is – standards don't strictly dictate development practice[3].

The role of SILs is best viewed as part of a process of deriving an appropriate development process from the hazard and risk analysis [26]. For example, consider IEC 61508, which applies to control systems meant to mitigate risks posed by equipment under control (e.g. an industrial metal fabrication tool) [21]. During hazard and risk assessment, engineers work out both the *consequences* (i.e. severity) and *likelihood* of harm (assuming no mitigation from the control system). These factors determine *risk*, which is compared with tolerable risk thresholds to determine the need for mitigation. From this, engineers derive a tolerable rate of failure for each mitigation function. This failure rate is converted into a SIL and used to drive development of appropriate development process. While other standards that use SIL do so in a broadly similar way, there are substantial differences and these can be a source of confusion [26].

### C. Partitioning and Integrity

Software running on one microprocessor frequently implements multiple functions. These functions might have different safety integrity needs and developers might want to save money by using less-rigourous processes to implement lower-SIL functions. However, software implementing one function could interfere with software implementing another, for example by writing to a random memory address. Standards typically address this by assigning to software the highest SIL of any function it implements *unless* developers demonstrate *partitioning integrity*. For example, IEC 61508 allows allocating different SILs to different parts of the software only when developers show that 'there is sufficient independence of implementation between these particular safety functions' [21].

### D. Adequate Confidence in Timing Claims

None of the common standards for software in safety critical applications clearly defines what constitutes adequate confi-

[3]Not even DO-178B [19] or DO-178C [20]. Developers and assessors agree a *Plan for Software Aspects of Certification* that details the specific activities undertaken to achieve the standard's SWDAL-specific objectives.

dence in a claim about task execution time [12]. For example, RTCA DO-178B requires 'review and analysis' to 'determine the correctness and consistency of the source code, including ... worst-case execution timing' [19]. This objective applies at all but the least-critical SILs. A companion document clarifies that dynamic approaches to determining WCET are sometimes appropriate but does not discuss which approaches are appropriate and not appropriate at each SIL [27].

### E. Survivability and Graceful Degradation

Common standards for software in safety critical applications focus mainly on preventing or detecting defects. But it is not generally possible to ensure that software-based systems will *never* fail. Recognising this, some standards, researchers, and authorities advocate building software so as to achieve *survivability* [28]. Survivability can be broadly defined as the ability of a system to provide *essential* services in the face of attacks and failures. To achieve this resilience, engineers build systems to reconfigure themselves in response to defined failure and attack conditions [11]. Such reconfiguration is *one* way to achieve the 'graceful degradation' that IEC 61508 recommends at low SILs and highly recommends at high SILs [21]. Researchers have formalised the definition of a survivability specification as $\{S, E, D, V, T, P\}$ where:

- $S$ specifies acceptable forms of service from the system (e.g. full service, limp-home mode)
- $E$ gives permitted values for each aspect of service value (e.g. $\{\text{mass air flow sensor} \mapsto \{\text{working}, \text{failed}\}, \ldots\}$)
- $D$ defines legal combinations of the values in $E$
- $V : S \times D \to \mathbb{N}_1$ gives the relative value that each configuration supplies under each environmental condition
- $T \subseteq S \times S \times D$ defines the legal mode transitions
- $P : S \to \{p : \mathbb{R} \mid 0 < p < 1\}$ specifies the probability with which the implementation of each service mode must meet its dependability requirements (e.g. $\{\text{full\_service} \mapsto 0.999, \text{limp\_home} \mapsto 0.99999\}$) [28]

Achieving graceful degradation always requires understanding how a system reacts to adverse events but does not always require explicit mode transitions. For example, graceful degradation might mean avoiding a design that causes a system that receives one too many requests to service none of them.

### F. Modes of Operation

Standards use the word 'mode' to mean different things. Each configuration in a survivability architecture can be called a mode. But when IEC 61508 directs engineers conducting hazard and risk analysis to 'give particular attention to abnormal or infrequent modes of operation of the [equipment under control]' [21], it means something different. The modes in this case are the ways in which the system is used (e.g. continuous production, prototyping/piecework, maintenance, etc.).

## V. Critique of Existing Problem Formulations

Most published work on MCS is from a scheduling perspective. In this section, we criticise the MCS problem formulations outlined in section II and suggest improvements.

### A. Impact of the WCET Confidence Monotonicity Assumption

As we showed in section III, it is not clear that the WCET confidence monotonicity assumption holds over all WCET assessment approaches. Static analysis approaches generally produce both greater confidence and greater overestimate than HWM testing approaches, but the issue is less clear where statistical approaches or hybrid approaches are concerned.

However, it might not matter that the WCET confidence monotonicity assumption does not hold universally. Suppose that, for a given task in a given system, a high-confidence WCET assessment approach yields a lower WCET limit than a low-confidence approach. Developers could simply use the high-confidence WCET limit for both $C_{\text{HI}}$ and $C_{\text{LO}}$.

When judging the risk associated with a design that includes MCS, developers need to reason about the confidence inspired by the WCET assessment approach used. This is complicated by our general inability to precisely quantify the total confidence / uncertainty associated with WCET assessment techniques. But this difficulty is not unique to MCS; it applies even when other scheduling approaches are used [12].

*1) Suggestion:* Confidence in WCET limit figures is incompletely understood and MCS might not provide benefit in cases where the WCET confidence monotonicity assumption does not hold. But there are important cases where it does, chief among them between HWM testing and static analysis. For the most critical functions, engineers and regulators might agree that only a technique that significantly overestimates true WCET provides sufficient confidence. Standard safety processes dictate that developers must either demonstrate partitioning integrity or use the same technique for all parts of the software. The MCS problem formulation should position MCS as a way to demonstrate partitioning integrity despite using less-conservative techniques such as HWM testing for functions that can tolerate greater uncertainty.

### B. Multiple Meanings of 'Criticality'

Vestal's formulation [1] used 'criticality' synonymously with SIL (or 'Design Assurance Level' in RTCA's DO-178B nomenclature [19]). That formulation uses criticality to indicate both the consequence of a task missing its deadline and confidence in the WCET limit figures used in timing analysis. As we noted in subsection IV-B, SIL is at best a crude indicator of these things. A high-SIL task overrunning its deadline might cause little or no safety impact and SIL is only the starting point for a negotiation between developers and assessors (if any) over which WCET assessment approach is appropriate. Moreover, engineers using a standard that defines five SILs are unlikely to use five different WCET assessment techniques.

The Baruah and Burns formulation [2], [5] extends the Vestal formulation [1] but uses the word criticality in much the same way. The Ekberg and Yi formulation [7] goes further, extending the Vestal formulation to combine MCS with reconfiguration for survivability. In doing so, it uses the word criticality as a label for what the survivability literature calls an 'acceptable form of service' [28]. This meaning is not interchangeable with either of the first two. Shifting to a new

form of service might increase the consequences of some tasks missing their deadlines and decrease the consequences for others. For example, suppose that a combat UAV is engaged in aerial combat and reconfigures into a dogfighting mode. In that mode, timely vision and control actuation might become more important than in normal flight while automatic de-icing becomes less important. But while the importance of a task might change when reconfiguring for survivability, confidence in WCET limits will not.

*1) Suggestion:* The MCS problem formulation should use different terms for each of the different concepts now being referred to as criticality. We suggest the following definitions:

- '*Importance*' should be used to describe the consequence of a task overrunning its deadline. Importance varies with service mode and might be lower than the SIL(s) associated with the tasks' service would indicate.
- '*Confidence*' should be used to describe confidence in a WCET limit or WCRT figure. *Uncertainty* is the lack of confidence (e.g. for the reasons discussed in section III).
- '*Mode*' should only be used as part of a complete term that describes a specific mode. *Service mode* should describe a mode used in reconfiguration for survivability. *Mode of operation* should describe the way in which a system is being used by its operators. *Scheduler mode* should be used to describe whether a scheduler is excluding low-importance tasks in order to help high-importance tasks meet their deadlines.

Because SILs are defined and used differently in different standards [26], the MCS problem should not be described in terms of SIL. Where helpful, documents giving advice for building systems using MCS can explain how importance and confidence map to a specific standard's definition of SIL.

### C. There Are Modes, and Then There Are Modes

The Ekberg and Yi MCS problem formulation [7] includes a single reconfiguration mechanism meant to facilitate both (a) surviving equipment failure and (b) recovering from an overrun of a low-confidence WCET figure. This is meant to promote generality (for its own sake) and reuse of validated approaches. They write, 'we would like the task model to be as general as possible. ... It is not unlikely that some existing solutions regarding the scheduling of regular mode switching systems can be adapted for mixed-criticality scheduling' [7]. However, it is not clear that (i) the Ekberg and Yi reconfiguration model is general enough for general survivability purposes, (ii) that a sufficiently general model would be suited to tolerating overruns, or (iii) that a combined model would be practical from a safety assurance standpoint.

In subsection IV-E we described a formal model of reconfiguration for survivability [28]. Such reconfigurations might take much longer than simply setting a mode variable to a different value. For example, a system migrating a task to a different computing node might need to load object code and initial state into that node's memory [29]. Clearly, the time budget for such reconfigurations cannot be included in the time budget for normal task execution. To accommodate such reconfigurations, reconfiguration architectures include special procedures that are executed to accomplish the reconfiguration [11].

As Baruah and Burns point out, MCS is meant to protect important services from a task failing to meet its low-confidence WCET limits due to a transient overload [5]. Achieving this requires reacting within a period. It is doubtful that one mechanism would be suitable for such reconfigurations *and* those that take many periods to complete.

As we described in subsection IV-C, many standards require developers to demonstrate temporal partitioning integrity. Where such integrity relies in part on MCS, demonstrating it requires demonstrating that reconfiguration protects important services from overruns of low-confidence WCET limits. Graceful degradation, on the other hand, is validated through architectural review, testing and analysis of the implementation, and analysis of how mode transitions could lead to hazards. Using a single mechanism to implement both mixes goals and concepts usually kept separate in the standards' process and might complicate the safety assurance effort. If tolerating overruns requires $x$ distinct modes and tolerating hardware failures requires $y$ distinct modes, a unified system might implement $x \times y$ modes and corresponding mode transitions. Because each of these must be tested, the combined implementation might significantly increase testing effort.

*1) Suggestion:* While reconfiguration in response to low-confidence WCET limit overruns and reconfiguration in response to hardware failure share some features, these reconfigurations are different and should be handled by separate reconfiguration mechanisms. However, as discussed in subsection V-B, some tasks might be more important in some service modes than in others. The MCS problem formulation should be given in terms of task importance rather than SIL to facilitate graceful degradation.

### D. To Kill or Not to Kill?

The *Partitioned Criticality* (PC) scheduling scheme assigns priorities so as to preserve temporal isolation without the need for a special monitoring and intervention mechanism [5]. In contrast, *Static Mixed Criticality* (SMC) uses run-time monitoring to restrict tasks to their run-time limits. *Adaptive Mixed Criticality* (AMC) goes further, using run-time monitoring to halt all criticality $\ell$ tasks whenever a criticality $\ell$ or higher task overruns its WCET limit. Unfortunately, some definitions of this monitoring are inappropriate for many applications.

It strains credulity to think that some tasks are so unimportant that they need never be executed following a transient overload. Indeed, anecdotes told to us in confidence reveal that some developers create high-SIL functions that depend on input from low-SIL functions, justifying this arrangement by designing the high-SIL task to tolerate bad or missing input until health monitoring restarts the low-SIL task. A reasonable MCS approach must restart halted tasks when it is safe to do so. Baruah and Burns do this when the processor is next idle [2]. The Ekberg and Yi MCS problem formulation kills unimportant tasks permanently, but a footnote raises the possibility of restarting halted tasks [7].

*1) Suggestion:* MCS problem formulations should explicitly define when they will restart tasks following an overrun. It would also help to explain what will happen when an important task overruns a high-confidence WCET limit.

## VI. Future Work: An MCS Assurance Argument

A *safety argument* for a given system explains how evidence in a *safety case* shows satisfaction of safety requirements and, ultimately, the operational definition of 'adequately safe to operate' in use in that system [30]. Other forms of *assurance argument*, such as *conformance argument*, can be used to show conformance with a safety standard [31].

Timing evidence, scheduling policy, and task allocation have a complex relationship with risk and safety standards. We have described some of that complexity in this paper. Such complexity makes it difficult to determine by ad hoc review whether a given formulation of the MCS problem captures the properties and evidence needed for safety assurance.

In prior work, we created an assurance argument to facilitate criticism of timing-related safety evidence and processes [12]. In this work, we suggested improvements to existing MCS problem formulations. We further suggest creating and criticising a safety argument for the revised formulation as a means of assessing its completeness and utility. Conformance arguments would likewise help to assess the problem formulation's fitness for use with a given standard and type of system.

## VII. Conclusions

In some safety-critical software systems, adequate risk reduction requires meeting deadlines. Unfortunately, some WCET assessment approaches that yield high confidence might substantially overestimate WCET. MCS promises to always permit important services to run for up to some high-confident system while delivering the utilisation that less-conservative WCET limits would allow.

In this paper, we reviewed three MCS problem formulations from a safety assurance perspective. We found four issues: (1) reliance on a questionable assumption about confidence in WCET limits, (2) use of the word 'criticality' to mean several different things, (3) flawed support for survivability-related reconfiguration, and (4) a haphazard treatment of recovery from transient overloads. We suggested improvements in the model formulation to address these issues. We further suggest the development of an assurance argument for MCS as a means of exploring the complex assurance issues and tradeoffs surrounding scheduling policy, task allocation, confidence in timing evidence, partitioning, and graceful degradation.

## Acknowledgement

We acknowledge the Swedish Foundation for Strategic Research (SSF) SYNOPSIS Project and EPSRC (UK) grant MCC (EP/K01 1626/1) for supporting this work.

## References

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. Int'l Real-Time Systems Symp. (RTSS)*, 2007.

[2] S. Baruah and A. Burns, "Implementing mixed criticality systems in Ada," in *Proc. Reliable Software Tech. (Ada-Europe)*, 2011.

[3] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *Proc. Int'l Real-Time Systems Symp. (RTSS)*, 2011.

[4] S. Baruah, "Certification-cognizant scheduling of tasks with pessimistic frequency specification," in *Proc. Int'l Symp. Industrial Embedded Systems (SIES)*, 2012.

[5] A. Burns and S. Baruah, "Timing faults and mixed criticality systems," in *Dependable and Historic Computing*, ser. LNCS. Springer Berlin Heidelberg, 2011, vol. 6875, pp. 147–166.

[6] A. Burns and R. Davis, "Mixed criticality systems: A review," Department of Computer Science, University of York, York, UK, Tech. Rep. MCC-1(b), July 2013.

[7] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-Time Syst.*, 2013, in press.

[8] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Improving the scheduling of certifiable mixed-criticality sporadic task systems," Uppsala University, Uppsala, Sweden, Tech. Rep., 2012.

[9] H.-M. Huang, C. Gill, and C. Lu, "Implementation and evaluation of mixed-criticality scheduling approaches for periodic tasks," in *Proc. Real-Time and Embedded Tech. and Applications Symp. (RTAS)*, 2012.

[10] M. Neukirchner, S. Stein, H. Schrom, J. Schlatow, and R. Ernst, "Contract-based dynamic task management for mixed-criticality systems," in *Proc. Int'l Symp. Industrial Embedded Systems (SIES)*, 2011.

[11] E. A. Strunk, "Reconfiguration assurance in embedded system software," Ph.D. dissertation, University of Virginia, Charlottesville, USA, 2005.

[12] P. Graydon and I. Bate, "Realistic safety cases for the timing of systems," *The Computer Journal*, 2013, in press.

[13] I. Bate, P. Conmy, T. Kelly, and J. McDermid, "Use of modern processors in safety-critical applications," *The Computer Journal*, vol. 44, no. 6, pp. 531–543, 2001.

[14] S. Edgar and A. Burns, "Statistical analysis of WCET for scheduling," in *Proc. Int'l Real-Time Systems Symp. (RTSS)*, 2001.

[15] R. Chapman, "Static timing analysis and program proof," DPhil Thesis, University of York, York, UK, 1995.

[16] S. Chattopadhyay and A. Roychoudhury, "Unified cache modeling for WCET analysis and layout optimizations," in *Proc. Int'l Real-Time Systems Symp. (RTSS)*, 2009.

[17] Rapita Systems, "RapiTime explained," http://www.rapitasystems.com/downloads/rapitime_explained_white_paper, July 2011.

[18] G. Bernat, A. Burns, and M. Newby, "Probabilistic timing analysis: An approach using copulas," *Journal of Embedded Computing*, vol. 1, no. 2, pp. 179–194, April 2005.

[19] RTCA DO-178B, *Software Considerations in Airborne Systems and Equipment Certification*. RTCA, Inc., 1992.

[20] RTCA DO-178C, *Software Considerations in Airborne Systems and Equipment Certification*. RTCA, Inc., 2011.

[21] IEC 61508:2010, *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*, 2nd ed. International Electrotechnical Commission, 2010.

[22] ISO 26262:2011, *Road Vehicles — Functional Safety*. International Organization for Standardization, 2011.

[23] CENELEC 50128:2001, *Railway Applications — Communications, Signalling and Processing Systems — Software for Railway Control and Protection Systems*. European Committee for Electrotechnical Standardization (CENELEC), 2001.

[24] Defence Standard 00-56, *Safety Management Requirements for Defence Systems, Issue 4*. UK Ministry of Defence, 2007.

[25] SAE ARP4754A, *Guidelines for Development of Civil Aircraft and Systems*. SAE, 2010.

[26] F. Redmill, "Understanding the use, misuse and abuse of safety integrity levels," Redmill Consultancy, Tech. Rep., 2005.

[27] RTCA DO-248B, *Final Report For Clarification Of DO-178B*. RTCA, Inc., 2001.

[28] J. C. Knight, E. A. Strunk, and K. J. Sullivan, "Towards a rigorous definition of information system survivability," in *Proc. DARPA Information Survivability Conf. and Expo.*, 2003.

[29] N. C. Audsley and M. Burke, "Distributed fault-tolerant avionic systems — A real-time perspective," in *Proc. IEEE Aerospace Conference*, 1998.

[30] T. P. Kelly, "Arguing safety — A systematic approach to managing safety cases," DPhil Thesis, University of York, York, UK, 1998.

[31] P. Graydon, I. Habli, R. Hawkins, T. Kelly, and J. Knight, "Arguing conformance," *IEEE Software*, vol. 29, no. 3, pp. 50–57, May 2012.