An Approach to Formally Specifying the Behaviour of Mixed-Criticality Systems

A. Burns ✉
University of York, York, UK

Cliff B. Jones ✉
Newcastle University, Newcastle upon Tyne, UK

Abstract

This paper proposes a formal framework for describing the relationship between a criticality-aware scheduler and a set of application tasks that are assigned different criticality levels. The exposition employs a series of examples starting with scheduling simple jobs and then moving on to mixed-criticality robust and resilient tasks. The proposed formalism extends the rely-guarantee approach, which facilitates formal reasoning about the functional behaviour of concurrent systems, to address *real-time* properties.

2012 ACM Subject Classification Computer systems organization → Embedded and cyber-physical systems; Software and its engineering → Real-time schedulability

Keywords and phrases real-time, scheduling, mixed criticality, rely/guaranteed conditions

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2022.12

Funding This research has been supported in part by EPSRC (UK) grants, STRATA and MCCps and by Leverhulme Trust grant RPG-2019-020

Acknowledgements The authors acknowledge useful suggestions made by Iain Bate, Sanjoy Baruah and Ian Hayes

1 Introduction

Since Vestal published his seminal paper in 2007 [61], there have been a wealth of models and protocols published [16, 17] on the topic of Mixed Criticality Systems (MCS). One of the aims of this wide ranging set of techniques is to improve the survivability of systems by providing a variety of degraded behaviours that can take effect if the system experiences overrunning execution times.

Inevitably these techniques require significant support from the underlying operating system. Unfortunately commercially-available, general-purpose, RTOSs do not provide this support. Hence, in order to utilise many of the more advanced scheduling ideas that are to be found in the MCS literature, it is necessary to develop the code for a bespoke scheduler as part of the application. Programming languages such as Ada [11] do provide the primitives necessary for this software to be developed but to deliver a reliable MCS scheduler the MCS protocols and models must be precisely specified. Research papers that focus on the algorithmic properties of protocols tend to give, at best, informal descriptions of the actual required run-time behaviour of the required scheduler.

The objective of the research described in this paper is *to develop a framework for formally specifying and reasoning about timing correctness properties of mixed-criticality systems*. The following paragraphs explain this objective in greater detail. In general, correctness in safety-critical systems can be considered from two perspectives: (i) (pre-run-time) verification, and (ii) (run-time) survivability.

Pre-run-time *verification* of a safety-critical system involves verifying, prior to deployment, that the run-time behaviour of the system will be consistent with expectations. Verification

34th Euromicro Conference on Real-Time Systems (ECRTS 2022).
Editor: Martina Maggio; Article No. 12; pp. 12:1–12:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

assumptions are made regarding the kinds of circumstances that will be encountered by the system during run-time and guarantees are used to specify the required runtime behaviour of the system (provided that the assumptions hold).

In contrast, *survivability* addresses expectations of system behaviour in the event that the assumptions fail to hold fully (in which case a fault or error is said to have occurred during run-time). Survivability may further be considered to comprise two notions: robustness and resilience [14]. Informally, the robustness of a system is a measure of the degree of fault it can tolerate without compromising the quality of service it offers; resilience refers to the degree of fault for which it can provide a degraded, yet acceptable, criticality-aware quality of service.

The contribution of this paper is to develop a framework for the formal specification of MCS; we define a formal approach that:

- Demonstrates that the Rely/Guarantee approach (see Section 2) can be extended to cover temporal properties (see Section 3) of concurrent systems (in addition to their functionality).
- Precisely specifies the required behaviour of a run-time scheduler (in normal and degraded modes of operation).
- Enables proofs to be developed and discharged that employ the contract(s) between the jobs and tasks comprising an application, and the scheduler.
- Enables, with additional specifications of the functional elements of the scheduler, the code of the scheduler to be produced as a refinement of these specifications.
- Enables the scheduler to be replaced or modified by verifying that a new version satisfies the original specification.
- Identifies the assumptions that the analysis (scheduling and execution time) makes such that the result of the analysis confirms that the system will meet its timing requirements.
- Enables the many approaches to resilience and robustness to be compared – this requires the formal framework to be sufficiently expressive to capture the semantics of the various schemes that have been proposed.

This initial description of our approach focusses on the specification aspects; future work will address verification. We do however demonstrate where proof can be used to ensure that, whenever a degraded mode must be entered, its prerequisites are ensured by the guaranteed conditions of the mode that has just been abandoned. We also make explicit the proof obligations on the offline scheduling analysis that must be applied to the application prior to deployment.

We explain the elements of the framework via a series of related, increasingly challenging, examples. The initial examples are sufficiently straightforward that, arguably, a full formal specification is not required; however the later examples do show the value of precise specifications. The examples illustrate the approach with at most two criticality levels, this helps to explain the framework, but again the full value of a formal approach comes when the system has increased complexity as happens when there are three or more criticality levels.

In this paper an MCS is assumed to consist of a finite set of jobs/tasks and a single specific Scheduler. Rely and guarantee conditions capture the run-time relationship between the Scheduler and the jobs/tasks, yielding a specification of the necessary behaviours/properties of the Scheduler. Note that this process does not delve into the internal structure of the Scheduler: the scheduling-theoretic issues of how it meets its specification (if indeed it can) is *not* the focus of this work. Rather, in this paper we are only seeking to provide a clear and intuitive explanation of the formalism. The history of formal methods (such as Hoare Logic) leads us to believe that methods can be developed for showing that specific MC-scheduling algorithms can satisfy (or not) the proof obligations that arise from the Rely/Guarantee (R/G) specifications. Related work in this area includes PROSA which addresses mechanised verification of results from scheduling analysis [21, 10]. (Mechanisation of R/G reasoning is on-going [29, 22]).

Organisation. The paper is organised as follows. After an introduction to R/G conditions (Section 2), the basic properties of the proposed framework are developed in Section 3 via a focus on *jobs* – this allows the approach to be motivated and explained. Mixed-criticality jobs are then covered in Section 4 including the introduction of fault-tolerance via modes of operation each with their own R/G conditions. Extensions of the same ideas to *tasks* are then given in Sections 5 and 6. Conclusions are in Section 7.

2 Introduction to Rely/Guarantee conditions

Hoare's 'Axiomatic Approach' provides the basis of a development method for sequential programs. Although [32] employed post conditions of single states, subsequent development methods such as VDM [39], B [1] and Event-B [2] use relational post conditions that define acceptable final states with respect to their initial values. Crucially, there is a relatively obvious notion of compositionality for sequential programs where a specification can be replaced by anything that satisfies its pre/post condition specification.

Finding compositional development methods for the development of concurrent programs proved to be difficult precisely because of the 'interference' that comes with (shared-variable) concurrency. One approach is to record and reason about interference using rely and guarantee conditions [37, 38] (a more algebraic presentation of the ideas is covered in [31]). The details and proof obligations of the R/G approach are not the main issue in the current paper. The basic idea is straightforward: just as pre conditions define a subset of possible starting states on which a component is expected to operate, *rely conditions* record interference that the specified component must tolerate; and, just as post conditions abstract from algorithms to achieve the transition from initial to final state, *guarantee conditions* are relations that define the maximum interference that the component may inflict on its environment. It is important to remember that pre and rely conditions are assumptions that a developer is invited to make; in contrast, guarantee and post conditions are obligations on the code to be created. A guarantee condition needs to be satisfied (only) as long as the corresponding rely condition is respected. Stating this negatively, if the environment makes a transition that does not satisfy the rely condition, the developed code is free from further obligations.

The R/G idea targeted the design of concurrent programs where the R/G conditions provide a way of decomposing designs. Papers such as [30, 42, 19] show that the R/G idea can be used to tackle the design of fault-tolerant CPS by using rely conditions to describe assumptions about physical system components. Where the physical components exhibit continuous change, the rely conditions record assumptions about the rate of such changes. This work also showed how layered R/G conditions can assist in addressing fault tolerance; *resilience* is represented by hierarchically related R/Gs—strong rely conditions address full functionality, weaker rely conditions are matched with lesser guarantees (perhaps only the safety-critical aspects), even weaker rely conditions might only guarantee safe fail-stop behaviour. *These properties of related R/G conditions are central to the framework developed in this paper.*

3 Job-based system model

This section focuses on a system comprising a set of jobs, $\mathcal{J}$, that are managed by a Scheduler (denoted by the symbol S). A representative job, $j \in \mathcal{J}$, has a relative deadline of D_j, arrives (and is released for execution) at time a_j and thus has an absolute deadline at time $d_j = a_j + D_j$. Let f_j denote the time at which it completes (finishes) its execution.[1] The set $act(\mathcal{J}, t)$ is the subset of $\mathcal{J}$ containing the jobs that are active at time t, i.e.

$$j \in act(\mathcal{J}, t) \Leftrightarrow j \in \mathcal{J} \wedge (a_j \leq t < f_j)$$

A job that is immediately terminated on arrival (as required in specific circumstances by some MCS protocols) has $f_j = a_j$; it is deemed never to be active and to have missed its deadline.

We assume a discrete time model in which all job parameters are given as non-negative rational numbers with arbitrary precision. Time is an external physical phenomenon: the Scheduler has no control over the passage of time.

The specification of each job, j, consists of its pre-condition, P_j, post-condition Q_j, rely condition R_j and guarantee condition G_j. In this paper each of these conditions is expressed as a predicate over the system state. For an actual system these conditions will capture both the functional and timing behaviour of the job; here we focus only on the temporal properties. This requires that system states are indexed by time[2] and that the rely and guarantee conditions directly reference time. We write $R_S(t)/G_S(t)$ for the Scheduler and $R_j(t)/G_j(t)$ for jobs.

Properties that should remain true as time progresses are normally classed as *invariants* but here are represented as rely or guarantee conditions. This is because the jobs (and Scheduler) must take action in order to maintain correct behaviour – a job will miss its deadline if it is not scheduled appropriately.

The primary concern for each job is its execution time; and hence we define, for each job j, $e_j(t)$ which is the amount of execution time the job has consumed up to time t. There are obvious properties (axioms) for e:

$$\forall j \in \mathcal{J}, t \bullet e_j(t) \leq \mathrm{WCET}_j \tag{1}$$

where WCET is the worst-case execution time of the job;

$$\forall j \in \mathcal{J}, t_1, t_2, t_1 < t_2 \bullet e_j(t_2) - e_j(t_1) \leq t_2 - t_1 \tag{2}$$

no job can execute faster than ‘real time’;

$$\forall j \in \mathcal{J}, t_1, t_2, t_1 < t_2 \bullet e_j(t_1) \leq e_j(t_2) \tag{3}$$

a job cannot ‘lose’ execution time; and

$$\forall j \in \mathcal{J} \bullet \Big(\forall t \leq a_j \bullet e_j(t) = 0 \ \wedge \ \forall t \geq f_j \bullet e_j(t) = e_j(f) \Big) \tag{4}$$

a job cannot execute before it arrives or after it has finished.

[1] A job that is yet to finish has $f{=}\infty$; a job that is permanently suspended but never terminated retains this value.

[2] A slightly different approach to handling the progress of time was taken in [40]. In that paper a distinction is made between an abstract notion of *Time* and the *ClockValues* stored in a computer.

In this section the scheduler is deemed to exist for the entire life-time of the system, it is therefore specified by a single rely condition $R_S(t)$ and a single guarantee condition $G_S(t)$.

The following derivations first illustrate the basic approach with a set of single criticality jobs. Note that the role of the formal framework is to represent precisely the relationship between the Scheduler and the client jobs in a range of degraded and partial behaviours. It is not a model of a particular scheduler's run-time behaviour; rather it is a specification of the required properties of any scheduler (and its schedulability test) that is being proposed for the particular problem under investigation.

A key feature of mixed-criticality models is that they allow a system to degrade gracefully when faults occur. This leads to the Scheduler's run-time behaviour having different modes of operation. In each mode, different R and G conditions for the jobs and scheduler are defined, as is the transition between R/G contracts.

We start by considering a finite set of jobs that each have the same criticality; there is no degraded behaviour and hence only a single mode of operation. A job j is characterised by its Worst-Case Execution Time, WCET_j (this is a value that will not be known with certainty) and C_j an estimate of WCET_j. The timely execution of a job *relies* on this estimate of WCET being valid, and the Scheduler can only meet its obligations with a *reliance* of each job executing for no more than C_j. If these rely conditions hold, a *valid* Scheduler *guarantees* to manage the processing capacity so as to ensure that all jobs complete by their deadlines regardless of when the jobs arrive; each job *guarantees* to execute, when active, for no more than C_j.

Note that the value C_j plays a number of roles: the job relies on its environment behaving according to whatever model or measuring process was used to derive C_j, but the job also has a contract with the scheduler not to execute for more than C_j. The scheduler is assumed to have used some form of analysis to verify (offline usually) that, if all jobs respect their guarantee conditions, then it will be able to provide the necessary capacity to each job. Hence the job can rely upon being allowed to execute for up to C_j before its deadline.

With all four axioms ((1)-(4) above) in force, the rely and guarantee conditions of any valid Scheduler are as follows:

$$R_S(t) \stackrel{\text{def}}{=} \forall j \in act(\mathcal{J}, t) \bullet e_j(t) \leq C_j$$

$$G_S(t) \stackrel{\text{def}}{=} \forall j \in act(\mathcal{J}, t) \bullet t + (C_j - e_j(t)) \leq d_j$$

The Scheduler relies on all jobs executing within their estimated WCET and guarantees to provide sufficient resource, following a defined policy, to ensure that each job always has sufficient space to complete before its deadline (i.e. that $t + (C_j - e_j(t)) \leq d_j$). [3] The Scheduler's guarantee is an *obligation* that must be achieved by its code – i.e. the Scheduler's offline schedulability test must ensure this property. The conditions $R_S(t)$ and $G_S(t)$ are defined to refer only to jobs that are active at time t.

In order to satisfy G_S, the Scheduler must manage the dispatching of jobs in an appropriate manner. If necessary it will allocate to each job up to C_j execution time. It follows that if $\text{WCET}_j \leq C_j$ then each job will terminate by its deadline (i.e. $f_j \leq d_j$).

The R and G conditions of each active job are therefore:

$$R_j(t) \stackrel{\text{def}}{=} \text{WCET}_j \leq C_j \wedge t + (C_j - e_j(t)) \leq d_j$$

[3] An alternative formulation [12] to the one presented here is for the Scheduler to guarantee a budget (of at least C for each job), and for each job to rely on this budget. Example specifications and further investigations indicated that the method defined in the current paper is the more realistic and effective.

$$G_j(t) \stackrel{\text{def}}{=} e_j(t) \leq C_j$$

At run-time, the job does not need to be aware of its deadline or current execution time; although more expressive and flexible behaviours may require this. Once a job (j) terminates the R_j and G_j conditions no longer apply.

The constraints imposed upon execution time are represented as guarantees and not post-conditions for a number of reasons:

1. post-conditions are, by definition, required to hold upon termination, but a failure may lead to the job not terminating;
2. to add fault tolerance (i.e. to cope with jobs whose estimated execution times are not respected) we will need to know the point in time at which a rely condition fails to hold (and hence a guarantee condition no longer has to hold); and
3. deadlines may change (or be removed) during the execution of the job (see later examples).

The semantics of rely/guarantee conditions is that guarantees are required to be met as long as the rely conditions are satisfied. If a job overruns and breaks its guarantee that $e_j(t) \leq C_j$ there must be a rely condition 'at fault'. For this reason, we explicitly include $\text{WCET}_j \leq C_j$ in the rely condition: in an environment where this assumption does not hold, a job is not obliged to guarantee its temporal properties.

If the environment (hardware platform including the influence of concurrently executing jobs, preemption effects on cache etc.) behaves such that the WCET estimate of some job k is exceeded, then this job may execute for more than C_k, thus breaking its guarantee condition. As a consequence the rely condition for the Scheduler would not be satisfied and hence it would be under no obligation to provide the necessary capacity to every job — some jobs may still be active at their deadlines. This takes us to the topic of survivability and how MCS supports graceful degradation.

4 Mixed-criticality jobs

To illustrate how a level of resilience can be added, two criticality levels are considered: HI-crit and LO-crit; with $\mathcal{J_L}$ a set of LO-crit jobs, $\mathcal{J_H}$ a set of HI-crit jobs, and $\mathcal{J} = \mathcal{J_L} \cup \mathcal{J_H}$. Job h is a representative HI-crit job; l is a representative LO-crit job; j continues to represent any job. So, for example, $R_h(t)$ is the rely condition for any HI-crit job, $h \in \mathcal{J_H}$. With Mixed-Criticality jobs there are two estimates of C_j: $C_j(L)$ and $C_j(H)$; with $C_j(L) \leq C_j(H)$ [61].

It is initially assumed that the system is either in the Normal mode, in which case all jobs should meet their deadlines, or in the HI-crit mode in which only the HI-crit jobs are guaranteed to meet their deadlines. For the Normal (N) mode the (R, G) conditions are as above except that $C_j(L)$ replaces C_j in R_j, G_j, R_S and G_S:

$$R_S^N(t) \stackrel{\text{def}}{=} \forall j \in act(\mathcal{J}, t) \bullet e_j(t) \leq C_j(L)$$

$$G_S^N(t) \stackrel{\text{def}}{=} \forall j \in act(\mathcal{J}, t) \bullet t + (C_j(L) - e_j(t)) \leq d_j$$

$$R_j^N(t) \stackrel{\text{def}}{=} \text{WCET}_j \leq C_j(L) \ \wedge \ t + (C_j(L) - e_j(t)) \leq d_j$$

$$G_j^N(t) \stackrel{\text{def}}{=} e_j(t) \leq C_j(L)$$

The rely and guarantee conditions for the N mode are therefore:

$$R^N(t) \ = \ R_S^N(t) \wedge \bigwedge_{j \in \mathcal{J}} R_j^N(t)$$

$$G^N(t) \;=\; G^N_S(t) \wedge \bigwedge_{j \in \mathcal{J}} G^N_j(t)$$

Most of these rely and guarantee conditions are mutually supportive in the sense that they "cancel out" when looking at the whole system. The only rely condition that depends on external compliance is:

$$\forall j \in \mathcal{J} \bullet \mathrm{WCET}_j \leq C_j(L)$$

4.1 Adding resilience to HI-crit jobs

Considering HI-crit jobs ($h \in \mathcal{J}_{\mathcal{H}}$) and their rely condition:

$$R^N_h(t) \stackrel{\text{def}}{=} \mathrm{WCET}_h \leq C_h(L) \;\wedge\; t + (C_h(L) - e_h(t)) \leq d_h$$

We want to give a higher (safer) bound on WCET, so we consider a more conservative value ($C_h(H)$), where $C_h(H) > C_h(L)$. Now for all HI-crit jobs (h) we have a new HI-crit mode (H) and:

$$R^H_h(t) \stackrel{\text{def}}{=} \mathrm{WCET}_h \leq C_h(H) \;\wedge\; t + (C_h(H) - e_h(t)) \leq d_h$$

$$G^H_h(t) \;\stackrel{\text{def}}{=}\; e_h(t) \leq C_h(H)$$

The Scheduler's definition for mode H is

$$R^H_S(t) \;\stackrel{\text{def}}{=}\; \forall h \in act(\mathcal{J}_{\mathcal{H}}, t) \bullet e_h(t) \leq C_h(H) \;\wedge\; \forall l \in act(\mathcal{J}_{\mathcal{L}}, t) \bullet e_l(t) \leq C_l(L)$$

$$G^H_S(t) \stackrel{\text{def}}{=} \forall h \in act(\mathcal{J}_{\mathcal{H}}, t) \bullet t + (C_h(H) - e_h(t)) \leq d_h$$

In this HI-crit mode there is no obligation to provide any level of service to the lower criticality jobs or indeed to prevent these jobs from using resources (perhaps at a background priority in a priority-based scheduler). Hence:

$$R^H_l(t) \;\stackrel{\text{def}}{=}\; \mathrm{WCET}_l \leq C_l(L)$$

$$G^H_l(t) \;\stackrel{\text{def}}{=}\; e_l(t) \leq C_l(L)$$

The above specification is, however, not sufficient for many of the protocols advocated for mixed-criticality scheduling. The standard 'mixed-criticality' mechanism for being able to add more capacity to the HI-crit jobs is to take computation time away from the LO-crit jobs. Or, more precisely, to no longer execute these jobs. This further adds to the guarantees of the Scheduler.

To facilitate this functionality it is necessary to know the time at which R^N_S became false (i.e. when an active HI-crit job has first executed for $C(L)$ without terminating). We refer to this as mode N's *deviation time*, η^N; defined by the following property:

$$\exists \eta^N, h \in act(\mathcal{J}_{\mathcal{H}}, \eta^N) \;\bullet\; e_h(\eta^N) \geq C_h(L) \;\wedge\; \forall t, t < \eta^N, g \in act(\mathcal{J}_{\mathcal{H}}, t) \;\bullet\; e_g(t) < C_g(L)$$

At the deviation time R^N_S becomes false, mode N is left and, simultaneously[4], mode H is entered. The rely and guarantee conditions $R^H(t)$ and $G^H(t)$ apply for $t \geq \eta^N$.

[4] The notion of simultaneous is taken from the Timebands [18] framework that allows instantaneous actions to be defined at one time band (granularity) but implemented by an activity at a finer time band.

We assume here the extreme Vestal behaviour of not executing *LO*-crit jobs again after η^N. This leads to a full specification for the guarantee condition for the Scheduler:

$$G_S^H(t) \stackrel{\text{def}}{=} \forall h \in act(\mathcal{J_H}, t) \bullet t + (C_h(H) - e_h(t)) \leq d_h \;\wedge\; \forall l \in act(\mathcal{J_L}, t) \;\bullet\; e_l(t) = e_l(\eta^N)$$

with a simplified rely condition as the Scheduler no longer relies on the behaviour of *LO*-crit jobs as it guarantees that they do not execute:

$$R_S^H(t) \stackrel{\text{def}}{=} \forall h \in act(\mathcal{J_H}, t) \bullet e_h(t) \leq C_h(H)$$

and therefore:

$$R^H(t) \;=\; R_S^H(t) \wedge \bigwedge_{l \in \mathcal{J_L}} R_l^H(t) \bigwedge_{h \in \mathcal{J_H}} R_h^H(t)$$

$$G^H(t) \;=\; G_S^H(t) \wedge \bigwedge_{l \in \mathcal{J_L}} G_l^H(t) \bigwedge_{h \in \mathcal{J_H}} G_h^H(t)$$

This strategy of pausing all *LO*-crit jobs is not an option that the Scheduler *could* choose, but a requirement that is part of the specification of the job's behaviour — and hence must be explicitly contained in G_S^H.

With this specification the *LO*-crit jobs are suspended; but they may execute later in another mode (perhaps after their deadlines). To abort these and future *LO*-crit jobs, rather than preempt them indefinitely, the Scheduler could (if specified to do so) enforce termination:

$$\forall t, t > \eta^N \;\bullet\; act(\mathcal{J_L}, t) = \emptyset$$

4.2 Transitioning from mode N to mode H

The specification above requires a movement from mode N to mode H. To provide useful fault tolerance, it must be true that, whenever the rely condition for N fails to be satisfied, the corresponding rely condition for H is satisfied (and remains so) i.e. at time η^N when $R^N(\eta^N)$ no longer pertains: $R^H(\eta^N)$ is satisfied. If $R^H(\eta^N)$ is true then the guarantee condition, $G^H(t)$, is delivered for all $t > \eta^N$, and as a consequence $R^H(t)$ must hold.

In general a mode change could involve modes with unrelated functionality and hence the truth of the rely condition in the new mode would need to be asserted independently of the rely condition in the old mode. This is identical to what is required at system startup where the rely condition of the initial mode must be established. In this work, however, we require a more constrained relationship between the modes:

▶ **Definition 1.** *Mode B is a* weakened *form of mode A if*
1. *for all times (t) before η^A when $R^A(t)$ is true then $R^B(t)$ is true (i.e. $R^A(t) \Rightarrow R^B(t)$); and*
2. *at time η^A when some aspect of $R^A(\eta^A)$ is no longer true $R^B(\eta^A)$ remains true.*

As $R^B(\eta^A)$ is true, it followed that $G^B(t)$ is true for all $t > \eta^A$.

Counter Example. We require that mode H is a weakening of mode N. Consider the first element of the definition of weakening: in two of the three rely conditions, this is indeed the case as:

$$R_S^N(t) \Rightarrow R_S^H(t); \quad R_l^N(t) \Rightarrow R_l^H(t)$$

but $R_h^N(t)$ does not have a simple relationship to $R_h^H(t)$. The first conjunct is a weakening of the 'external' rely condition as $\text{WCET}_h \leq C_h(L) \Rightarrow \text{WCET}_h \leq C_h(H)$. The second conjunct is, however, a strengthening; hence modes N and H do not have the required hierarchical relationship – H is not a weakened form of N.

A Modified Definition of Mode N (N^*). In order to assert that mode H is a weakened form of the initial mode it is necessary to constrain the behaviour of the Scheduler further in the Normal mode. It must do more than simply guarantee to provide for all jobs $C(L)$ before the deadline d, it must also reserve sufficient slack so that, at any time a switch can be made, it is possible to guarantee $C(H)$ before d.

It follows that, for a HI-crit jobs, h, to be schedulable in both N^* and H modes, there exists a virtual deadline v_h with

$$v_h \leq d_h - (C_h(H) - C_h(L))$$

that is defined (and confirmed) by the applicable scheduling analysis, such that: if the Scheduler in mode N^* guarantees $C(L)$ by v, then the Scheduler in mode H will be able to guarantee $C(H)$ by d.[5] To accommodate this constraint the guarantee condition of the Scheduler in mode N^* must be modified to:

$$G_S^{N^*}(t) \stackrel{\text{def}}{=} \forall j \in act(\mathcal{J}, t) \bullet t + (C_j(L) - e_j(t)) \leq v_j$$

and the Rely conditions of HI-crit jobs becomes

$$R_h^{N^*}(t) \stackrel{\text{def}}{=} \text{WCET}_h \leq C_h(L) \ \wedge \ t + (C_h(L) - e_h(t)) \leq v_h$$

For LO-crit jobs (l) $v_l = d_l$ and hence G_S^N has not changed for these jobs. For HI-crit jobs (h) there is a proof obligation on the scheduling analysis to demonstrate:

$$\forall t, h \in act(\mathcal{J}_\mathcal{H}, t) \bullet G_S^{N^*}(t) \Rightarrow t + (C_h(H) - e_h(t)) \leq d_h \tag{5}$$

Such an obligation could be verified using mechanised proof tools such as PROSA [21, 10].

▶ **Lemma 2.** *Mode H is a weakening of mode N^*.*

Proof. As noted above $\forall t : R_S^N(t) \Rightarrow R_S^H(t)$ and $R_l^N(t) \Rightarrow R_l^H(t)$. The modification to N^* does not effect these rely conditions. Also $\text{WCET}_h \leq C_h(L) \Rightarrow \text{WCET}_h \leq C_h(H)$ (as $C_h(H) \geq C_h(L)$). Finally $t + (C_h(L) - e_h(t)) \leq v_h \Rightarrow t + (C_h(H) - e_h(t)) \leq d_h$ as $v_h \leq d_h - (C_h(H) - C_h(L))$.

The second step is to show that, at time η^{N^*} (when $R^{N^*}(\eta^{N^*})$ fails), $R^H(\eta^{N^*})$ remains true. Condition $R^{N^*}(\eta^{N^*})$ is false because the $WCET$, for some HI-crit job k, is not bounded by $C_k(L)$. Moreover η^{N^*} is the first time instant at which R^{N^*} is false. Hence at time η^{N^*}, $R_k^{N^*}(\eta^{N^*})$ is false, but $R_k^H(\eta^{N^*})$ is true as $C_k(H) > C_k(L)$.[6] ◀

This weakening property and the proof obligation represented by eqn (5) are therefore sufficient to ensure that, whenever the Normal mode must be abandoned, the HI-crit mode can be entered and will deliver its guaranteed behaviour. The final point to note about the transition from N^* to H is that the Guarantee conditions are also weakened. The system moves from guaranteeing all job deadlines to just guaranteeing the HI-crit ones. Hence $G^{N^*}(t) \Rightarrow G^H(t)$.

[5] This virtual deadline is used directly in the EDF-based scheduling scheme EDF-VD [5] and in fixed-priority scheduling is equivalent to the worst-case (maximum) computed response time of the HI-crit job in the Normal mode [6]. Note whatever scheduling protocol is employed at run-time there is an implicit (if not explicit) virtual deadline in the Normal mode. If this were not the case then there would be insufficient spare capacity in the Normal mode to satisfy the extra demand of the HI-crit mode.

[6] Strictly, we require $C_k(H) > C_k(L) + \delta$ where δ is the minimum time step that the system can undertake in its discrete model of time.

4.3 Postponing the deviation time

As noted in the introduction, the main focus of this paper is to motivate and define a formal framework for the specification of mixed criticality systems. In this section we are able to give an example of how this framework can be utilised.

A system is considered to degrade at deviation time η^{N^*} which is defined, above, as the first time that a HI-crit job executes beyond its $C(L)$ constraint. But if this deviation time could be postponed then the dynamics of the system may alleviate the need to make the mode change – the LO-crit jobs could continue to meet their deadlines. Possible favourable dynamic behaviours include sporadic jobs not arriving at their maximum rate, and other jobs executing for less than their maximum $C(L)$ bound. To explore the possibility of delaying the deviation time consider again the specification of the N^* mode:

$$R_S^{N^*}(t) \stackrel{\text{def}}{=} \forall j \in act(\mathcal{J}, t) \bullet e_j(t) \leq C_j(L)$$

$$G_S^{N^*}(t) \stackrel{\text{def}}{=} \forall j \in act(\mathcal{J}, t) \bullet t + (C_j(L) - e_j(t)) \leq v_j$$

$$R_j^{N^*}(t) \stackrel{\text{def}}{=} \text{WCET}_j \leq C_j(L) \ \wedge \ t + (C_j(L) - e_j(t)) \leq v_j$$

$$G_j^{N^*}(t) \stackrel{\text{def}}{=} e_j(t) \leq C_j(L)$$

where $v_j = d_j$ for LO-crit jobs and $v_l \leq d_l - (C_l(H) - C_l(L))$ for HI-crit jobs.

If all jobs behave according to this R/G specification then all virtual deadlines will be met. This implies there is a weakened form of behaviour (which we denote as mode $\hat{N}^*$):

$$R_S^{\hat{N}^*}(t) \stackrel{\text{def}}{=} \forall j \in act(\mathcal{J}, t) \bullet t \leq v_j$$

$$G_j^{\hat{N}^*}(t) \stackrel{\text{def}}{=} t \leq v_j$$

with $G_S^{\hat{N}^*} = G_S^{N^*}$ and $R_j^{\hat{N}^*} = R_j^{N^*}$.

From the definition of the virtual deadline we have $R_S^{N^*} \Rightarrow \hat{R}_S^{N^*}$ and $G_j^{N^*} \Rightarrow \hat{G}_j^{N^*}$.

The deviation time (when $\hat{R}_S^{N^*}$ becomes false for the first time) is now when a HI-crit job is still executing at its virtual deadline. And this time is likely to be significantly later than that provided by the earlier definition. Note also that this alternative definition of the deviation time for the normal mode changes what needs to be monitored – from execution time to elapsed time. This is likely to reduce the runtime overheads of the MCS scheduler.

Again it is straightforward to prove that mode H is a weakening of (the modified) mode $\hat{N}^*$, and the proof obligation on the offline scheduling analysis (eqn (5)) must again be used to validate the v values assigned to each HI-crit job. Recent scheduling results [8] have demonstrated that for fixed priority-based scheduling and AMC-rtb analysis the same v values are valid for the original definition of deviation time and the one derived in this section. That paper also demonstrated the benefits in terms of run-time performance that is gained from postponing the mode change.

The proposed framework allowed this new protocol to be easily defined and verified. Further properties can be proven (such as the above definition of deviation time being the latest possible). In this introductory paper, however, priority is given to extending the framework to task-based systems.

5 Task-based system model

The above treatment of mixed-criticality jobs has demonstrated that the proposed specification framework has sufficient expressive power to capture the properties commonly required of job-based systems. The scheduling literature typically describes jobs as being organised within tasks — in this section we extend the study to cope with tasks.

A real-time system is deemed to consist of a set of tasks. A single execution of the code of a task is a job. So a task gives rise to a sequence of jobs. The scheduler determines the order in which jobs from different tasks are executed. With a task-based model there is an assumption that the duration of the system is unbounded. This means that any specification framework must cater for the return of the system from any degraded mode back to the initial mode for the system (and to allow these mode changes to occur numerous times). We assume that each task k delivers a potentially unbounded sequence of jobs, k^1, k^2 etc, with job k^m having arrival time a_k^m and completion time f_k^m. This sequence is not 'reset' as new modes are entered; it continues to extend indefinitely.

This treatment focuses on issues related to execution time and mixed criticality. It does not directly address the rely and guarantee conditions related to when and how a task is released for execution. For example, time-triggered tasks require their job releases to be guaranteed by some Dispatcher; and event-triggered tasks rely on their releasing events obeying some minimum separation requirement. These issues are covered here by each task guaranteeing that its jobs do not arrive too early — a rely condition for the Scheduler.

The system is again assumed to be defined over two criticality levels, LO-crit and HI-crit, and to have two modes of behaviour: N^* and H. We however drop, for ease of presentation, the superscript from N in the following. To define a general model, each of the defining temporal parameters of each task (D, T, C, V) has an L and a H value.

We again make use of sets: $\mathcal{T}$ is the set of all tasks, $\mathcal{T_L}$ the set of LO-crit tasks, and $\mathcal{T_H}$ the set of HI-crit tasks, and $\mathcal{T} = \mathcal{T_L} \cup \mathcal{T_H}$. The axioms defined in Section 3 still apply.

At any time t, each task k has a single *current job*. We let $c(t)$ be the index of this job (for ease of presentation, we just use c for this index as the t value is always implied). Hence the current job of task k is denoted by k^c. This job may have finished, but the next job of this task has not yet arrived ($f_k^c \leq t < a_k^{c+1}$). In task models that allow a job to arrive before the previous job of the same task has finished (i.e. tasks with $D > T$), the 'current' job is the one that arrived first.

We modify the definition of 'active' to cater for tasks; a task is active if its current job has not yet terminated:

$$k \in act(\mathcal{T}, t) \Leftrightarrow k \in \mathcal{T} \wedge (a_k^c \leq t < f_k^c)$$

In each of the criticality modes the relative parameters (V_k and D_k) are added to the arrival time a_k^c to give the absolute values: v_k^c, $d_k^c(L)$ and $d_k^c(H)$.

5.1 Vestal-inspired example

This section specifies the required behaviour of the system (Scheduler and tasks) for a typical model inspired by the Vestal approach [61]. The properties of this model are, briefly:

- System starts in the mode N in which all jobs of all tasks execute for no more than $C(L)$ and all job deadlines are met.
- All LO-crit tasks are assumed (or constrained) to execute for no more than $C(L)$.
- All HI-crit tasks are assumed (or constrained) to execute for no more than $C(H)$.
- If any, or indeed all, HI-crit tasks execute for more than $C(L)$ then:

  - all HI-crit tasks must still meet their deadlines;
  - all LO-crit tasks have their periods and deadlines increased, but must still meet their deadlines.
- If there is an idle instant then the system must return to the Normal mode of operation.

This extension of the Vestal model is often referred to as the *elastic task model* [20] in which the periods and deadlines of LO-crit tasks are extended from $T_l(L)$ (and $D_l(L)$) to $T_l(H)$ (and $D_l(H)$), but are still guaranteed.

The major difference when moving from jobs to tasks is that each task, like the Scheduler, exists for the full duration of the time spent in each mode. Although individual jobs terminate, the task does not (in the model being utilised here). So $R_k(t)$ and $G_k(t)$ are the rely and guarantee conditions of task k, but they refer to the job that is current (and possibly active) at time t.

For the Vestal-inspired model outlined above we have, for all LO-crit tasks, $l \in \mathcal{T_L}$, $C_l(L) = C_l(H)$, $T_l(H) > T_l(L)$, $D_l(H) > D_l(L)$ and $V_l = D_l(L)$ and for all HI-crit tasks, $h \in \mathcal{T_H}$, $C_h(L) < C_h(H)$, $T_h(L) = T_h(H)$, $D_h(L) = D_h(H)$ and $V_h < D_h(L) - (C_h(H) - C_H(L))$.

The conditions for the normal mode N are:

$$R_S^N(t) \stackrel{\text{def}}{=} \forall k \in act(\mathcal{T}, t) \bullet e_k^c(t) \leq C_k(L) \ \wedge (c > 1 \Rightarrow a_k^c - a_k^{c-1} \geq T_k(L))$$

$$G_S^N(t) \stackrel{\text{def}}{=} \forall k \in act(\mathcal{T}, t) \bullet t + (C_k(L) - e_k^c(t)) \leq v_k^c$$

$$R_k^N(t) \stackrel{\text{def}}{=} \text{WCET}_k \leq C_k(L) \ \wedge \ k \in act(\mathcal{T}, t) \Rightarrow t + (C_k(L) - e_k^c(t)) \leq v_k^c(L)$$

$$G_k^N(t) \stackrel{\text{def}}{=} e_k^c(t) \leq C_k(L) \ \wedge \ (c > 1 \Rightarrow \ a_k^c - a_k^{c-1} \geq T_k(L))$$

R_S^N contains the separation condition: if the current job is not the first instantiation of the task then it must arrive at least $T_k(L)$ after the previous job.

In the HI-crit mode, H, we have a similar formulation but with different parameters:

$$R_S^H(t) \stackrel{\text{def}}{=} \forall k \in act(\mathcal{T}, t) \bullet e_k^c(t) \leq C_k(H) \ \wedge (c > 1 \Rightarrow a_k^c - a_k^{c-1} \geq T_k(H))$$

$$G_S^H(t) \stackrel{\text{def}}{=} \forall k \in act(\mathcal{T}, t) \bullet t + (C_k(H) - e_k^c(t)) \leq d_k^c(H)$$

$$R_k^H(t) \stackrel{\text{def}}{=} \text{WCET}_k \leq C_k(H) \ \wedge \ k \in act(\mathcal{T}, t) \Rightarrow t + (C_k(H) - e_k^c(t)) \leq d_k^c(H)$$

$$G_k^H(t) \stackrel{\text{def}}{=} e_k^c(t) \leq C_k(H) \ \wedge \ (c > 1 \Rightarrow \ a_k^c - a_k^{c-1} \geq T_k(H))$$

These two formulations can easily be combined into a single specification that is a function of the mode (N or H) but are separated here to improve readability.

5.2 Transitioning from N to H

In this and the following section we consider the movement between modes; from Normal, N, to the HI-crit mode, H, and then the return to the Normal mode. In a long-lived task-based system there may be many such transitions between N and H. Each time a mode is entered, we consider this to be a new *occurrence* of the mode and therefore there is a new *occurrence* of the Scheduler for that mode. A move from N to H involves one occurrence of the N-mode Scheduler terminating and, instantaneously, a new occurrence of the H-mode Scheduler starting its execution[7]. A natural linkage between Scheduler occurrences is for the

[7] An implementation may utilise a single Scheduler that modifies its behaviour depending upon which mode is current. Nevertheless, from a modelling point of view we consider each occurrence of the Scheduler to be a distinct execution.

post-condition of one mode, say A (Q_S^A), to ensure the pre-condition of the follow-on mode, B (P_S^B), with $Q_S^A \Rightarrow P_S^B$.

We note that the two mode changes contained within this task-based two-level mixed criticality system are of a quite different nature. The movement from N to H is forced, as N must be left. But the transition from H back to N is one of preference – both modes are acceptable, but the functional behaviour of the system is enhanced by being in the N mode.

In Section 4.2 we noted that as mode N is left at time η^N, due to $R^N(\eta^N)$ being false, we must prove that $R^H(\eta^N)$ is true. This involves two steps. First, at any time $t < \eta^N$, $R^N(t) \Rightarrow R^H(t)$. Second, at time η^N, when $R^N(\eta^N)$ is broken, $R^H(\eta^N)$ remains true.

Following the approach in Section 4.2, the task model has again made use of a virtual deadline for HI-crit jobs; from this we derive the proof obligation:

$$\forall t, h \in act(\mathcal{T_H}, t) \bullet G_S^N(t) \Rightarrow t + (C_h(H) - e_h^c(t)) \leq d_h^c(H) \tag{6}$$

Counter Example. With this Vestal-inspired example, the periods of the LO-crit tasks are expanded when the H mode is entered. It is therefore **not** true that $a_l^c - a_l^{c-1} \geq T_l(L) \Rightarrow a_l^c - a_l^{c-1} \geq T_l(H)$ as $T_l(H) > T_l(L)$. Hence R_S^N does **not** imply R_S^H.

A Modified Definition of Mode H (H^*). We must again modify the specification. However on this occasion rather than strengthen the rely condition in mode N we weaken the definition of the rely condition for the Scheduler in the HI-crit mode:

$$R_S^{H^*}(t) \overset{\text{def}}{=} \forall k \in act(\mathcal{T}, t) \bullet e_k^c(t) \leq C_k(H) \wedge (c > 1 \wedge t > \eta^N \Rightarrow a_k^c - a_k^{c-1} \geq T_k(H))$$

Note the addition of $t > \eta^N$, the constraint on the arrival times of jobs in the new mode only applies strictly *after* η^N. The Guarantee condition of mode H^* is unchanged ($G^{H^*}(t) = G^H(t)$) and for the tasks: $R_k^{H^*}(t) = R_k^H(t)$, and $G_k^{H^*}(t) = G_k^H(t)$.

▶ **Lemma 3.** *Mode H^* is a weakening of mode N.*

Proof. First, $\forall t < \eta^N$: For LO-crit tasks: $C_l(H) = C_l(L)$ and $v_l^c = d_l^c$ hence $R_l^N = R_l^H$ (so $R_l^N \Rightarrow R_l^H$). For HI-crit tasks: $\text{WCET}_h \leq C_h(L) \Rightarrow \text{WCET}_h \leq C_h(H)$ (as $C_h(H) \geq C_h(L)$); and $t+(C_h(L)-e_h^c(t)) \leq v_h^c \Rightarrow t+(C_h(H)-e_h^c(t)) \leq d_h^c$ as $v_h^c \leq d_h^c-(C_h(H)-C_h(L))$. Hence $R_h^N \Rightarrow R_h^{H^*}$. For the Scheduler, the first conjunct is appropriate as $e_k^c(t) \leq C_k(L) \Rightarrow e_k^c(t) \leq C_k(H)$, the second conjunct does not apply as $t < \eta^N$.

The second step (showing R^{H^*} is true at time η^N) follows the proof of Lemma 2; noting again that the second conjunct of $R_S^{H^*}(t)$ does not apply when $t = \eta^N$. ◀

As $R^{H^*}(\eta^N)$ is true, it follows that $G^{H^*}(t)$ is true for all $t > \eta^N$ and hence $R^{H^*}(t)$ is true for all $t \geq \eta^N$ as long as all task execution times are bounded by $C_k(H)$.

The proof obligations on the necessary scheduling analysis must allow for all LO-crit generated jobs to arrive at the time of the mode change. One of the advantages of this more formal specification of the Scheduler's behaviour is that it helps identify this constraint explicitly. We note that many examples of published scheduling algorithms for mixed-criticality systems (for example [15]) do allow LO-crit jobs to arrive (and subsequently execute) at the time of the mode change even if that would not be allowed in the new mode. However this property is often hidden within the analysis (by the use of a 'floor plus one' rather than a 'ceiling' representation of job arrivals). Within our formal framework the property is explicit.

To summarise, in order to prove that R^H is true whenever a forced mode change can occur, we note three distinct situations:

1. Conjuncts within R^H are weakened forms of those in R^N and remain true.

2. Conjuncts in R^N must be strengthened so that they then imply the corresponding conjunctions in R^H.
3. Conjuncts in R^H must be weakened so that they are implied by the corresponding conjunctions in R^L.

The above example makes use of all three strategies.

5.3 Transitioning from H to N

As long as the execution times of the HI-crit tasks are bounded by their $C(H)$ estimates, the system will stay in the H mode. All the rely conditions will remain true. However it is desirable to return to the Normal mode if possible as this mode provides a better level of service – i.e. LO-crit tasks will be able to occur more often and have shorter deadlines.

Once the over-running HI-crit job that caused the transition to mode H has terminated, there is the possibility that all new jobs can be released with their LO-crit parameters and, if they all execute for no more than $C(L)$, all deadlines can be met. But we know that any scheduling scheme can only guarantee deadlines if there is bounded (indeed often zero) residual work in the system at the time the Normal mode is (re-)activated [7]. It is therefore scheduler specific as to when the system is 'safe' to return to the Normal N mode of operation.

May/must constraints [19] are useful here. If the system is idle (there are no jobs to execute), it is usual to state that the scheduler *must* return the system to the Normal mode, but it *may* make this change earlier if a proof obligation has shown that such a transition is safe.

In terms of the framework presented in this paper a switch back to N mode is allowed only when the scheduling obligations (as represented by G_S^N) of that mode can be satisfied by the current Scheduler. If these obligations are satisfied, the move from H to N can be sanctioned by an appropriate pre-condition on the Normal mode. An example of one such pre-condition is the commonly used protocol that the Normal mode can only be (re-)entered at time t if there are no active jobs at time t (other than ones that arrive at time t):

$$P_S^N(t) \stackrel{\text{def}}{=} k \in act(\mathcal{T}, t) \Rightarrow a_k^c = t$$

The Scheduler for the Normal mode can therefore assume this property and it is the responsibility of the Scheduler in the HI-crit mode to enforce it whenever it invokes a mode change back to Normal. In other words this is a post-condition for the Scheduler in mode H:

$$Q_S^H(t) \stackrel{\text{def}}{=} k \in act(\mathcal{T}, t) \Rightarrow a_k^c = t$$

6 Robustness and resilience

Here we extend the treatment for tasks to show how we can more systematically specify levels of robustness and resilience for mixed-criticality systems, the motivation here being to develop a means of quantifying robustness and resilience. The first step in this process is to specify the various schemes being proposed.

Informal definitions of robustness and resilience are provided in [14] – i.e. the robustness of a system is a measure of the level of faults it can tolerate without compromising the quality of service it offers; resilience, by contrast, refers to the level of faults for which it can provide degraded yet acceptable (e.g. safe) quality of service. It is noted in [14] that there are a number of standard responses in the fault tolerance literature for systems that suffer transient faults (equating to one or more concurrent job failures in this work):

1. Fail (Fully) Operational – all tasks/jobs execute correctly (i.e. meet their deadlines).
2. Fail Robust – some tasks are allowed to skip a job but all non-skipped jobs execute correctly and complete by their deadlines; the quality of service at all criticality levels is unaffected by job skipping. Many periodic control tasks have this property [62]; there is sufficient inertia in the physical system to allow the occasional control signal to be missed.
3. Fail Resilient – some lower criticality tasks are given reduced service such as having their periods/deadlines extended, priorities dropped and/or their execution budgets reduced; if the budget is reduced to zero then this is equivalent to subsequent jobs of the task being abandoned.
4. Fail Safe/Restart – where the level of failure exceeds what Fail Resilient bounds can accommodate, more extreme responses are required including rebooting or system shutdown (if the application has a fail-safe state). If a fail-safe state cannot be achieved then the system may need to rely on best-effort tactics that have no guarantees. This is, of course, the last resort to achieving survivability.

6.1 Failure modes

The framework developed above has been extended to include a number of more complex behaviours that arise from supporting robust and resilient behaviour. In this section we briefly outline a set of possible failure modes.

Fail operational – FO. A HI-crit job experiences a fault if it executes for more than $C(L)$. One measure of Fail Operational is therefore the number of such job failures that can be accommodated while still meeting all task deadlines. However, if a job from a HI-crit task executes for more than $C(L)$, we still assume that the $C(H)$ bound remains operational.

One criticism of those models derived from Vestal [61] is that they usually assume that any overrun of $C(L)$ results in an execution time of $C(H)$. In practice this is very unlikely to occur, a minor overrun is more likely. We therefore introduce a parameter, C_O, that represents a unit of overrun (for all jobs). Fail Operational is a measure of how many such overruns can be accommodated. Let O denote this number over all the tasks. A HI-crit job that executes for more than $C_h(L)$ but less than $C_h(L) + C_O$ has an O value of 1. In general, a task has an O value of n if its overrun is between $(n-1) * C_O$ and $n * C_O$.

The metric for Fail Operational is therefore the maximum O value allowed (F_O) in a defined interval, I_O. This interval could be of a fixed length (and would usually be much greater than the maximum task period). Alternatively it could be the interval from the current time back to when there was an idle moment, m, defined by:

$$\exists m, m < t \;\bullet \Big(\forall k \in act(\mathcal{T}, m) \;\bullet\; a_k^c = m\Big) \wedge$$
$$\forall n, m < n < t \bullet (\exists k \;\bullet\; k \in act(\mathcal{T}, n) \wedge a_k^c < n)\Big)$$

so the only active tasks at time m are those that released a job at that time, and there are active tasks that have not just been released for all times between m and t. Note m must exist as system startup (time 0) matches the definition of m as the only active tasks are those released at time 0. We note that m is a function of t, hence $m(t)$ in the following.

To compute O at time t, we need to know how many overruns each job has experienced. This can be computed as follows:

$$O \;=\; \sum_{\forall h \in \mathcal{T_H}, s \;\bullet\; t > a_h^s \geq m(t)} \left\lceil \frac{e_h^s(t) - C_h(L)}{C_O} \right\rceil_0$$

where $\lceil\rceil_0$ constrains the ceiling function to return a value no less than 0.

If this value is greater than 1 but no greater than F_O then the system mode should be Fail Operational (FO) with all tasks meeting their deadlines. It follows that the rely and guarantee conditions for the Scheduler are as follows. Remember that for LO-crit tasks $C(H) = C(L)$:

$$R_S^{FO}(t) \stackrel{\text{def}}{=} \forall k \in act(\mathcal{T}, t) \bullet e_k^c(t) \le C_k(H) \wedge (a_k^c = t \wedge c > 1) \Rightarrow a_k^c - a_k^{c-1} \ge T_k(L) \wedge$$

$$\sum_{\forall h \in \mathcal{T_H}, s \bullet t > a_h^s \ge m(t)} \left\lceil \frac{e_h^k(t) - C_h(L)}{C_O} \right\rceil_0 \le F_O$$

$$G_S^{FO}(t) \stackrel{\text{def}}{=} \forall k \in act(\mathcal{T}, t) \bullet t + C_k(L) - e_k^c(t) \le v_k^c \wedge$$

$$\forall h \in \mathcal{T_H} \bullet a_h^c \ge m(t) \wedge e_h^c(t) > C_h(L) \Rightarrow t + C_k(H) - e_k^c(t) \le d_k^c(H)$$

As there are no overruns in the normal mode we can deduce that $R_S^N \Rightarrow R_S^{FO}$.

Note this formulation is structurally different from that given earlier for a pure Vestal-like model. What the Scheduler must rely on is a property of the whole set of HI-crit tasks, not a specific property of each individual task. The Scheduler can therefore guarantee $C_h(H)$ (by the task's deadline) to any HI-crit tasks that overrun. But this guarantee is subject to the rely condition remaining true (i.e. there is a bound on the number and extent of these overruns).

The specification of the HI- and LO-crit tasks in the normal mode, and for most tasks in the FO mode, is simply

$$R_k^{FO}(t) \stackrel{\text{def}}{=} \text{WCET}_k \le C_k(L) \wedge k \in act(\mathcal{T}, t) \Rightarrow t + C_k(L) - e_k^c(t) \le v_k^c$$

$$G_k^{FO}(t) \stackrel{\text{def}}{=} e_k^c(t) \le C_k(L) \wedge (a_k^c = t \wedge c > 1) \Rightarrow a_k^c - a_k^{c-1} \ge T_k(L)$$

But for the tasks that overrun, they experience a mode change that moves the system to a variant of FO:

$$R_h^{FO^*}(t) \stackrel{\text{def}}{=} \text{WCET}_h \le C_h(H) \wedge h \in act(\mathcal{T_H}, t) \Rightarrow t + C_h(H) - e_h^c(t) \le d_h^c(H)$$

$$G_h^{FO^*}(t) \stackrel{\text{def}}{=} e_h^c(t) \le C_h(H) \wedge (a_h^c = t \wedge c > 1) \Rightarrow a_h^c - a_h^{c-1} \ge T_h(H)$$

For the non overrunning tasks and the Scheduler $R^{FO^*} = R^{FO}$, and $G^{FO^*} = G^{FO}$.

A small number of tasks experiencing this change will not cause the Scheduler to change mode, unless its rely condition is invalidated. The proof obligation (6) will again ensure that $R_h^{FO^*}$ is a weakening of R_h^N and R_h^{FO}.

In summary, a system stays in the normal mode until a single HI-crit task executes for more than $C(L)$. The system then moves to mode FO with the overrunning task behaving according to mode FO^*. Further HI-crit tasks may overrun and move to mode FO^*. Eventually either an idle instant occurs and the system will return to the normal mode N, or the F_O count is breached and R_S^{FO} is invalidated. The system will now fail unless there is a further degraded mode it can transition to; such a mode is considered next.

Fail robust – FR. A *robust task* is one that can safely drop one non-started job in a defined time interval. Each task (be it HI-crit or LO-crit), as part of its definition, has a robustness parameter, w. If a task has successfully completed the execution of w consecutive jobs then the Scheduler can drop the next job (before it has been given any execution time). As such jobs should only be dropped if they have to be, this requires a new mode: FR (Fail Robust). This mode will only be entered if the rely condition of the Scheduler in mode

FO becomes false (i.e there are more than F_O overruns). Within FR F_R overruns will be tolerated (with $F_R > F_O$); i.e.

$$\sum_{\forall h \in \mathcal{T}_{\mathcal{H}}, s \ \bullet \ t > a_h^s \geq m(t)} \left\lceil \frac{e_h^s(t) - C_h(L)}{C_O} \right\rceil_0 \leq F_R$$

We introduce a predicate, $req_k(t)$ (short for required) that returns true if the current job of task k at time t must be executed. Tasks that require all their jobs to execute are assigned, for ease of presentation, $w = 0$. The conditions for the current job (k^c) of task k to be required are: (1) $w_k = 0$, or (2) the task has not yet executed w_k jobs, i.e. $c \leq w$, or (3) one of the previous w_k jobs (before c) had a zero execution time — this is an indication that the job was dropped. This leads to the following definition:

$$req_k(t) \stackrel{\text{def}}{=} w_k = 0 \ \vee \ c \leq w_k \ \vee \ \exists s, s \in c - w_k..c-1 \ \bullet \ e_k^s(f_k^s) = 0$$

In other words, $req_j(t)$ is false only when the last w_j jobs of τ_j (i.e. $j_j^{c-1}, j_j^{c-2}, \ldots j_j^{c-w_j}$) have completed successfully. A non robust task is always 'required' (in that its current job must always complete). The R/G conditions can again be easily derived for the Fail Robust mode:

$$R_S^{FR}(t) \stackrel{\text{def}}{=} \forall k \in act(\mathcal{T}, t) \ \bullet \ e_k^c(t) \leq C_k(H) \ \wedge (a_k^c = t \wedge c > 1) \Rightarrow a_k^c - a_k^{c-1} \geq T_k(L) \ \wedge$$
$$\sum_{\forall h \in \mathcal{T}_{\mathcal{H}}, s \ \bullet \ t > a_h^s \geq m(t)} \left\lceil \frac{e_h^s(t) - C_h(L)}{C_O} \right\rceil_0 \leq F_R$$

Note this is a weakening of the rely condition as $R_S^{FO} \Rightarrow R_S^{FR}$ which follows from $F_R > F_O$ i.e. more overruns can be tolerated in the Fail Robust mode.

We can now complete the full specification. The Scheduler only guarantees execution time to those jobs that are required; moreover, if a job is not required the Scheduler ensures it does not execute.

$$G_S^{FR}(t) \stackrel{\text{def}}{=} \forall k \in act(\mathcal{T}, t) \ \bullet \ req_k(t) \Rightarrow t + C_k(L) - e_k^c(t) \leq v_k^c \ \wedge$$
$$\forall k \in \mathcal{T} \ \bullet \ a_k^c = t \wedge \neg req_k(t) \Rightarrow f_k^c = t \ \wedge$$
$$\forall h \in \mathcal{T}_{\mathcal{H}} \ \bullet \ a_h^c \geq m(t) \wedge e_h^k(t) > C_h(L) \Rightarrow t + C_k(H) - e_k^c(t) \leq d_k^c(H)$$

The tasks only need execution time if they are required; their guarantee conditions remain true even if the current job does not execute.

$$R_k^{FR}(t) \stackrel{\text{def}}{=} \text{WCET}_k \leq C_k(L) \ \wedge \ k \in act(\mathcal{T}, t) \wedge req_k(t) \Rightarrow t + C_k(L) - e_k^c(t) \leq v_k^c$$
$$G_k^{FR}(t) \stackrel{\text{def}}{=} e_k^c(t) \leq C_k(L) \ \wedge \ (a_k^c = t \wedge c > 1) \Rightarrow a_k^c - a_k^{c-1} \geq T_k(L)$$

As with mode FO, an individual HI-crit task can fail (rely condition becomes invalid, false) leading to a weakened specification:

$$R_h^{FR^*}(t) \stackrel{\text{def}}{=} \text{WCET}_h \leq C_h(H) \ \wedge \ h \in act(\mathcal{T}_{\mathcal{H}}, t) \Rightarrow t + C_h(H) - e_h^c(t) \leq d_h^c(H)$$
$$G_h^{FR^*}(t) \stackrel{\text{def}}{=} e_h^c(t) \leq C_h(H) \ \wedge \ (a_h^c = t \wedge c > 1) \Rightarrow a_h^c - a_h^{c-1} \geq T_h(H)$$

Note the req_k condition has been removed from the rely condition as the current job must be required to execute as it has a non-zero execution time (i.e. a value that exceeded $C(L)$); also this is another weakening of the rely condition.

Again with this specification the Scheduler must rely on a property of the whole set of HI-crit tasks, not a specific property of each individual task.

Fail resilient (graceful degradation) – GD. Once the count of job failures becomes greater than F_R, the FR mode must be abandoned as the rely condition of the Scheduler becomes false. To add resilience, a number of different general strategies for graceful degradation have been discussed in the literature [55, 45, 54]. Some strategies are hierarchical, in that they form a natural progression of increasingly severe forms of degradation that are invoked by increasingly severe forms of failure. Others take the form of alternative approaches.

All strategies are defined by their level of fault tolerance (the maximum O count they can deal with) and their impact on LO-crit tasks. Example strategies include:

1. Increasing the periods and deadlines of LO-crit tasks [60, 59, 36, 58, 57, 53, 25], called *task stretching*, the *elastic task model* or *multi-rate* (also see Section 5.1)
2. Imposing only a weakly-hard constraint on the LO-crit tasks [24, 51]
3. Decreasing the computation times of the LO-crit tasks [13, 4], perhaps by utilising an imprecise mixed-criticality (IMC) model [50, 52, 49, 33] or budget control [26, 27]
4. Moving some LO-crit tasks to a different processor that has not experienced a criticality mode change [63, 64, 35, 3].
5. Abandoning LO-crit work in a disciplined sequence [23, 34, 28, 56, 46, 47].

Some example strategies have already been described in the paper. Of course the specific set of schemes that may be applicable will depend on the details of the application. Nevertheless, any collection of approaches can be (partially) ordered using preferences and the strengths/weaknesses of the rely conditions of the Scheduler.

In general, the full set of modes forms a lattice with the Normal N mode at the top, and the Fail Safe (FS) mode at the bottom (see below). Preferences are assigned to reflect the structure of this lattice (N is the most preferred mode, FS the least). The least preferred resilient mode is the one that represents the total abandonment of all LO-crit jobs. We define this to be the backstop mode (BM). In the following BM is entered after the failure of GD:

$$R_S^{BM}(t) \stackrel{\text{def}}{=} \forall h \in act(\mathcal{T_H}, t) \bullet e_h^c(t) \le C_h(H) \wedge (a_h^c = t \wedge c > 1) \Rightarrow a_h^c - a_h^{c-1} \ge T_h(H)$$

$$G_S^{BM}(t) \stackrel{\text{def}}{=} \forall h \in act(\mathcal{T_H}, t) \bullet t + C_h(H) - e_h^c(t) \le d_h^c \wedge \forall l \in act(\mathcal{T_L}, t) \bullet e_h^c(t) = e_h^c(\eta^{GD})$$

where again η^{GD} is the time this mode is entered (i.e. when some graceful degradation mode, GD must be abandoned). Now no active LO-crit jobs execute.

$$R_h^{BM}(t) \stackrel{\text{def}}{=} \text{WCET}_h \le C_h(H) \wedge h \in act(\mathcal{T_H}, t) \Rightarrow t + C_h(H) - e_h^c(t) \le d_h^c(H)$$

$$G_h^{BM}(t) \stackrel{\text{def}}{=} e_j^h(t) \le C_h(H) \wedge (a_h^c = t \wedge c > 1) \Rightarrow a_h^c - a_h^{c-1} \ge T_h(H)$$

$$R_l^{BM}(t) \stackrel{\text{def}}{=} true$$

$$G_l^{BM}(t) \stackrel{\text{def}}{=} (a_l^c = t) \Rightarrow (f_l^c = t)$$

hence any newly arrived LO-crit job is immediately finished (aborted).

Fail safe/restarts – FS. The final 'strategy' is fail safe, perhaps via fail stop, followed by a subsequent restart (which may use a cold, warm or hot standby). It is not the purpose of this paper to review these approaches to fault tolerance. But for completeness we note that wherever possible there should be a mode (FS) which guarantees a fail safe outcome.

$$P_S^{FS} \stackrel{\text{def}}{=} true$$

$$R_S^{FS}(t) \stackrel{\text{def}}{=} \mathit{true}$$

$$G_S^{FS}(t) \stackrel{\text{def}}{=} t \leq (\eta^{BM} + D_S^{FS})$$

$$Q_S^{FS} \stackrel{\text{def}}{=} \mathit{safe\_shut\_down}$$

where D_S^{FS} is the (relative) deadline of the scheduler in this mode – there is a bound on how far t can reach.

This mode must be the lowest preference mode (i.e. be at the base of the lattice). It can always be entered, but must only be entered when all Schedulers in other modes have rely conditions that are false. Note we give the Scheduler a deadline in this mode to instigate the shut-down activity, but no further functional information can be given as the Scheduler is no longer operational.

6.2 Robust and resilient mode changes

In the above discussion a number of Scheduler modes have been introduced. They naturally form a sequence based on preference; the inverse of this sequence describes the behaviour of the system as it experiences graceful degradation:

$$N \to FO \to FR \to GD \to BM \to FS$$

An application could have a number of intermediate modes between FR and BM. In addition there could be a number of 'best-effort' (not guaranteed) behaviours/modes between BM and FS.

For the set of operational modes it will be necessary to show they form a hierarchy:

$$R^N \Rightarrow R^{FO} \Rightarrow R^{FR} \Rightarrow R^{GD} \Rightarrow R^{BM} \Rightarrow R^{FS}$$

Moreover, at the time a rely condition becomes invalid and the next mode is entered (at times η^N, η^{FO}, η^{FR},η^{BM}), it can be proven (see Lemmas 2 and 3) that the new rely condition is true and henceforth the guarantee condition holds.

In contrast to this gradual decline in functionality, a system that is programmed to recover will move directly from any of the degraded modes back to mode N. This move is driven by preference; but to reenter the Normal mode there will be some prerequisites. As noted in Section 5.3 this could be simply that at the time the Normal mode is re-entered there are no active tasks that had been released prior to this time.

7 Conclusions and Future Work

There is extensive published work on Mixed-Criticality scheduling and implementation, but not on their formal specification. We believe formalisation is essential since the notion of mixed criticality has subtle semantics: often concepts such as correctness, resilience and robustness are neither straightforward nor intuitive for such systems. The R/G approach has proved a successful formalism for specifying non-real-time safety-critical systems and our main contribution in this paper is to extend R/G to (i) time, and (ii) multiple criticalities.

The proposed framework is based on an ordering of modes (in general, this would form a lattice) with the normal mode (N) being at the top and a Fail Stop (FS) mode at the base. Each mode has an R/G coupling with a move down the ordering accompanied by a weakening of the rely and guarantee conditions. Examples were used to show that to obtain a true hierarchical relationship between the rely conditions (e.g. $R^A \Rightarrow R^B$, for modes A and B), it is often necessary to strengthen the R^A and/or weaken the R^B conditions. A movement of the system down the ordering (from mode A to B) occurs only when forced by R^A no longer being true. At this time it is necessary to prove that R^B remains true. The return of the system back to mode N is sanctioned by the rely and pre conditions of N being reestablished.

The examples presented in this paper have demonstrated that the developed approach has the expressive power necessary to enable a wide range of possible runtime strategies to be precisely specified and evaluated (in terms of their internal consistency). Further work will address the application of the R/G specifications in the development of the necessary run-time code that will be needed to support these mixed-criticality protocols. This would benefit from mechanical proof support as undertaken by the PROSA team [21, 10]. Although this work is not covered in the current paper there is ample evidence that R/G specifications can form the basis for the formal development of implementations. A useful example is tackled in [43, 41]: although not scheduling per se, Simpson's 4-slot algorithm is a delicate piece of intricate code for asynchronous communication mechanisms. A number of other examples of developments based on R/G specifications are listed and/or tackled in [48, 31, 44, 9].

References

1. J.-R. Abrial. *The B-Book: Assigning programs to meanings.* Cambridge University Press, 1996.
2. J.-R. Abrial. *The Event-B Book.* Cambridge University Press, Cambridge, UK, 2010.
3. J. Baik and K. Kang. Schedulability analysis for task migration under multiple mixed-criticality systems. In *Proc Korean Society of Computer Science*, page X, 2019.
4. S. K. Baruah, A. Burns, and Z. Guo. Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviours. In *Proc. ECRTS*, pages 131–140, 2016.
5. S.K. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. In *Proc. of the 19th Annual European Symposium on Algorithms (ESA 2011) LNCS 6942, Saarbruecken, Germany*, pages 555–566, 2011.
6. S.K. Baruah, A. Burns, and R.I. Davis. Response-time analysis for mixed criticality systems. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, pages 34–43, 2011.
7. I. Bate, A. Burns, and R.I. Davis. An enhanced bailout protocol for mixed criticality embedded software. *IEEE Transactions on Software Engineering*, 43(4):298–320, 2016.
8. I. Bate, A. Burns, and R.I. Davis. Analysis-runtime co-design for adaptive mixed criticality scheduling. In *Proc. of forthcoming IEEE RTAS, Pre publication version privately communicated.*, 2022.
9. R. Bornat and H. Amjad. Explanation of two non-blocking shared-variable communication algorithms. *Formal Aspects of Computing*, 25(6):893–931, 2013.
10. S. Bozhko and B.B. Brandenburg. Abstract response-time analysis: A formal foundation for the busy-window principle. In Marcus Völp, editor, *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, volume 165 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:24, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
11. A. Burns. Why the expressive power of programming languages such as Ada is needed for future cyber physical systems. In *Ada-Europe International Conference on Reliable Software Technologies*, pages 3–11. Springer, 2016.
12. A. Burns, S. Baruah, C.B. Jones, and I. Bate. Reasoning about the relationship between the scheduler and mixed-criticality jobs. In *Proc. 7th Int. RTSS Workshop On Mixed Criticality Systems (WMC)*, pages 17–22, 2019.

13 A. Burns and S.K. Baruah. Towards a more practical model for mixed criticality systems. In *Proc. 1st Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 1–6, 2013.

14 A. Burns, R. Davis, S. K. Baruah, and I. Bate. Robust mixed-criticality systems. *IEEE Transactions on Computers*, 67(10):1478–1491, 2018.

15 A. Burns and R.I. Davis. Response-time analysis for mixed-criticality systems with arbitrary deadlines. In *Proc. Workshop on Mixed Criticality Systems (WMC)*, pages 13–18, 2017.

16 A. Burns and R.I. Davis. A survey of research into mixed criticality systems. *ACM Computer Surveys*, 50(6):1–37, 2017.

17 A. Burns and R.I. Davis. Mixed criticality systems: A review (13th edition). Technical Report MCC-1(13), available at https://www-users.cs.york.ac.uk/~burns/review.pdf and the White Rose Repository, Department of Computer Science, University of York, 2022.

18 A. Burns and I.J. Hayes. A timeband framework for modelling real-time systems. *Real-Time Systems Journal*, 45(1–2):106–142, June 2010.

19 A. Burns, I.J. Hayes, and C.B. Jones. Deriving specifications of control programs for cyber physical systems. *Computer Journal*, 63(5):774–790, 2020.

20 G. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *IEEE Real-Time Systems Symposium*, pages 286–295, 1998.

21 F. Cerqueira, F. Stutz, and B.B. Brandenburg. PROSA: A case for readable mechanized schedulability analysis. In *Proc. 28th Euromicro Conference on Real-Time Systems (ECRTS)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 273–284, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

22 Diego Machado Dias. *Mechanising an algebraic rely-guarantee refinement calculus*. PhD thesis, School of Computing, Newcastle University, 2017.

23 T. Fleming and A. Burns. Incorporating the notion of importance into mixed criticality systems. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 33–38, 2014.

24 O. Gettings, S. Quinton, and R.I. Davis. Mixed criticality systems with weakly-hard constraints. In *Proc. International Conference on Real-Time Networks and Systems (RTNS))*, pages 237–246, 2015.

25 C. Gill, J. Orr, and S. Harris. Supporting graceful degradation through elasticity in mixed-criticality federated scheduling. In Jing Li and Zhishan Guo, editors, *Proc. 6th Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 19–24, 2018.

26 X. Gu and A. Easwaran. Dynamic budget management with service guarantees for mixed-criticality systems. In *Proc. Real-Time Systems Symposium (RTSS)*, pages 47–56. IEEE, 2016.

27 X. Gu and A. Easwaran. Dynamic budget management and budget reclamation for mixed-criticality systems. *Real-Time Systems*, 55:552–597, 2019.

28 X. Gu, K.-M. Phan, A. Easwaran, and I. Shin. Resource efficient isolation mechanisms in mixed-criticality scheduling. In *Proc. 27th ECRTS*, pages 13–24. IEEE, 2015.

29 I. J. Hayes. Generalised rely-guarantee concurrency: An algebraic foundation. *Formal Aspects of Computing*, 28(6):1057–1078, 11 2016.

30 I.J. Hayes, M. Jackson, and C.B. Jones. Determining the specification of a control system from that of its environment. In Keijiro Araki, Stefani Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, volume 2805 of *LNCS*, pages 154–169. Springer Verlag, 2003.

31 I.J. Hayes and C.B. Jones. A guide to rely/guarantee thinking. In Jonathan Bowen, Zhiming Liu, and Zili Zhan, editors, *Engineering Trustworthy Software Systems – Third International School, SETSS 2017*, volume 11174 of *LNCS*, pages 1–38. Springer, 2018.

32 C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

33 L. Huang, I-H. Hou, S.S. Sapatnekar, and J. Hu. Graceful degradation of low-criticality tasks in multiprocessor dual-criticality systems. In *Proc. of the 26th International Conference on Real-Time Networks and Systems*, RTNS, pages 159–169. ACM, 2018.

34 P. Huang, P. Kumar, N. Stoimenov, and L. Thiele. Interference constraint graph: A new specification for mixed-criticality systems. In *Proc. 18th Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2013.

35 S. Iacovelli, R. Kirner, and C. Menon. ATMP: An adaptive tolerance-based mixed-criticality protocol for multi-core systems. In *Proc. IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*, pages 1–9, 2018.

36 M. Jan, L. Zaourar, and M. Pitel. Maximizing the execution rate of low criticality tasks in mixed criticality system. In *Proc. 1st WMC, RTSS*, pages 43–48, 2013.

37 C.B. Jones. *Development Methods for Computer Programs including a Notion of Interference*. PhD thesis, Oxford University, June 1981. Printed as: Programming Research Group, Technical Monograph 25.

38 C.B. Jones. Specification and design of (parallel) programs. In *Proc. of IFIP*, pages 321–332. North-Holland, 1983.

39 C.B. Jones. *Systematic Software Development using VDM*. Prentice Hall International, second edition, 1990. URL: `http://homepages.cs.ncl.ac.uk/cliff.jones/ftp-stuff/Jones1990.pdf`.

40 C.B. Jones and A. Burns. A rely-guarantee specification of mixed-criticality scheduling. In Valentin Cassano and Nazareno Aguirre, editors, *Mathematical Foundations of Software Engineering: Essays in Honor of Tom Maibaum on the Occasion of his Retirement*, Tribute Series. College Publications, 2022.

41 C.B. Jones and I.J. Hayes. Possible values: Exploring a concept for concurrency. *Journal of Logical and Algebraic Methods in Programming*, 85(5):972–984, 2016.

42 C.B. Jones, I.J. Hayes, and M.A. Jackson. Deriving specifications for systems that are connected to the physical world. In Cliff B. Jones, Zhiming Liu, and Jim Woodcock, editors, *Formal Methods and Hybrid Real-Time Systems*, volume 4700 of *Lecture Notes in Computer Science*, pages 364–390. Springer Verlag, 2007.

43 C.B. Jones and K.G. Pierce. Elucidating concurrent algorithms via layers of abstraction and reification. *Formal Aspects of Computing*, 23(3):289–306, 2011.

44 C.B. Jones and N. Yatapanage. Investigating the limits of rely/guarantee relations based on a concurrent garbage collector example. *Formal Aspects of Computing*, 31(3):353–374, 2019. on-line April 2018.

45 J.C. Laprie. Dependable computing and fault tolerance: Concepts and terminology. In *Digest of Papers, The Fifteenth Annual International Symposium on Fault-Tolerant Computing*, pages 2–11, Michigan, USA, 1985.

46 J. Lee, H.S. Chwa, L.T.X. Phan, I. Shin, and I. Lee. MC-ADAPT: Adaptive task dropping in mixed-criticality scheduling. *ACM Trans. Embed. Comput. Syst.*, 16:163:1–163:21, 2017.

47 J. Lee and J. Lee. Mc-flex: Flexible mixed-criticality real-time scheduling by task-level mode switch. *IEEE Transactions on Computers*, page online, 2021. `doi:10.1109/TC.2021.3111743`.

48 Hongjin Liang, Xinyu Feng, and Ming Fu. A rely-guarantee-based simulation for verifying concurrent program transformations. In *Proc. 39th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, pages 455–468, New York, NY, USA, 2012.

49 D. Liu, N. Guan, J. Spasic, G. Chen, S. Liu, T. Stefanov, and W. Yi. Scheduling analysis of imprecise mixed-criticality real-time tasks. *IEEE Transactions on Computers*, 67(7):975–991, July 2018.

50 D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi. EDF-VD scheduling of mixed-criticality systems with degraded quality guarantees. In *Proc. IEEE RTSS*, pages 35–46, 2016.

51 R. Medina, E. Borde, and L. Pautet. Directed acyclic graph scheduling for mixed-criticality systems. In Johann Blieberger and Markus Bader, editors, *Reliable Software Technologies – Ada-Europe*, pages 217–232. Springer International Publishing, 2017.

52 R.M. Pathan. Improving the quality-of-service for scheduling mixed-criticality systems on multiprocessors. In Marko Bertogna, editor, *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*, volume 76 of *Leibniz International Proc. in Informatics (LIPIcs)*, pages 19:1–19:22. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

53 S. Ramanathan, A. Easwaran, and H. Cho. Multi-rate fluid scheduling of mixed-criticality systems on multiprocessors. *Real-Time Systems*, 54:247–277, 2018.

54 B. Randell, J-C. Laprie, H. Kopetz, and B. Littlewood(Eds.). *Predictably Dependable Computing Systems*. Springer, 1995.

55 B. Randell, P.A. Lee, and P.C. Treleaven. Reliability issues in computing system design. *ACM Computing Surveys*, 10(2):123–165, 1978.

56 J. Ren and L.T.X. Phan. Mixed-criticality scheduling on multiprocessors using task grouping. In *Proc. 27th ECRTS*, pages 25–36. IEEE, 2015.

57 H. Su, P. Deng, D. Zhu, and Q. Zhu. Fixed-priority dual-rate mixed-criticality systems: Schedulability analysis and performance optimization. In *Proc. Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 59–68. IEEE, 2016.

58 H. Su, N. Guan, and D. Zhu. Service guarantee exploration for mixed-criticality systems. In *Proc. Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–10. IEEE, 2014.

59 H. Su and D. Zhu. An elastic mixed-criticality task model and its scheduling algorithm. In *Proc. of the Conference on Design, Automation and Test in Europe*, DATE, pages 147–152, 2013.

60 H. Su, D. Zhu, and D. Mosse. Scheduling algorithms for elastic mixed-criticality tasks in multicore systems. In *Proc. RTCSA*, 2013.

61 S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.

62 N. Vreman, A. Cervin, and M. Maggio. Stability and Performance Analysis of Control Systems Subject to Bursts of Deadline Misses. In Björn B. Brandenburg, editor, *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*, volume 196 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:23, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

63 H. Xu and A. Burns. Semi-partitioned model for dual-core mixed criticality system. In *23rd International Conference on Real-Time Networks and Systems (RTNS 2015)*, pages 257–266, 2015.

64 H. Xu and A. Burns. A semi-partitioned model for mixed criticality systems. *Journal of Systems and Software*, 150:51 – 63, 2019.