Realistic Safety Cases for the Timing of Systems

Patrick Graydon[1,2] and Iain Bate[1,2]

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

Timing is often seen as the most important property of systems after function, and safety-critical systems are no exception. In this paper, we consider how timing is typically treated in safety assurance and in particular the safety arguments being proposed by industry and academia. A critique of these arguments is performed based on how systems are generally developed and how evidence is gathered. Significant weaknesses are exposed resulting in a more appropriate safety argument being proposed. As part of this work techniques for identifying relationships, in the form of contracts, between parts of the argument and the strength of evidence are used. The work is demonstrated using a Computer Assisted Braking example, specifically an Anti-Lock Braking System for a car, as it is a classic example of a component that may be used "Out of Context", as discussed in a number of safety standards, and may also be reused across a number of systems as well as part of a product line.

Keywords: Safety Arguments, Real-Time, Scheduling, Testing, Analysis, Fault Tolerance

Received 00 TODO 2012

1. INTRODUCTION

Many safety-critical systems are 'hard real-time' systems in which failure to meet a timing requirement might be catastrophic [1]. Typically, the principal timing-related safety evidence is timing analysis. Developers determine the Worst-Case Execution Time (WCET) of each task, compute the Worst-Case Response Times (WCRT) of the task set, and then use the results to show that timing requirements will be met (barring hardware failure). Unfortunately, this method does not always inspire sufficient confidence. Moreover, this problem is worsening as microprocessors grow more complex. Accordingly, developers use health monitoring (i.e. detection of a fault condition followed by an appropriate response) to mitigate occurrences that might otherwise lead to failures [2, 3].

Judging whether a system's timing-related hazards have been adequately managed requires understanding both the quality of the timing analysis and how health monitoring complements it. Our thesis is that it is possible to construct a software safety argument that better conveys the crucial timing-related information that is inadequately conveyed by current standards-based and argument-based approaches. We achieve this by presenting a more accurate description of the evidence, the limitations of the evidence, and the relationship between timing analysis and health monitoring. In this paper, we make four contributions:

1. An assessment of the state of the art of WCET determination and WCRT analysis
2. A discussion of current prescriptive and argument-based approaches to safety assurance
3. Identification of issues hindering assessment of confidence that timing requirements are met
4. An example illustrating how to argue more compellingly that a system adequately manages its timing-related hazards

We discuss the state of the art in WCET estimation and WCRT analysis in Section 2. We discuss the state of the practice in safety assurance in Section 3. We present our example argument in Section 4. Finally, we conclude in Section 5.

2. THE STATE OF THE ART OF WCET ESTIMATION, WCRT ANALYSIS, AND HEALTH MONITORING

The standard approach to ensuring that a system meets its timing requirements is to determine the WCET of its tasks and then compute WCRT. Unfortunately, on modern platforms, determining WCET is problematic. Moreover, WCRT analysis typically requires making and compensating for simplifying assumptions that are false. These difficulties limit the confidence inspired by timing analysis. Health monitoring can restore some of

this confidence. However, the confidence inspired by the combination of timing analysis and health monitoring is often unclear.

2.1. The Difficulty of Accurately Determining WCET

Typical approaches to analysing WCRT, such as Park et al.'s seminal work, require each task's exact WCET as input [4]. Unfortunately, it is not generally possible to determine the exact WCET of anything other than the simplest of problems. Instead, developers produce either a (safe, pessimistic) *upper bound* on WCET or a (possibly optimistic) *estimate* of WCET. Approaches to determining WCET can be divided into three categories: (a) *static* (analysis-based) approaches; (b) *dynamic* (measurement-based) approaches; and (c) *hybrid approaches*.

Ideally, an approach would satisfy two criteria. First, underestimation of WCET should be adequately unlikely. Timing analysis based on an underestimate could falsely conclude that a system meets timing requirements. Accordingly, the probability of underestimation must be small enough to yield adequate confidence that deadlines will be met. Second, because large overestimates waste processor resources, the approach should not grossly overestimate WCET. Unfortunately, no current approaches always meets both of these criteria for all target processors. Current static approaches work for some moderately complex single-core processors but find their limits in more complex architectures (e.g., multicore). Most measurement-based approaches might underestimate WCET with an unknown probability. Moreover, it is possible to err using any of these approaches. Achieving adequate confidence in the WCET figure requires developers to consider and address each form of potential error.

2.1.1. Static Approaches to Determining WCET

Static analysis of WCET considers, in principle, all possible paths through the analysed code. While a suite of timing tests might not include the worst case, static analysis does. However, despite this tremendous advantage, static analysis does not always produce perfect confidence in a usefully-tight upper bound on WCET in all applications. Moreover, there are possibilities for error that developers must address.

In general, computing an exact WCET from program text is intractable and would require solving the unsolvable halting problem [5]. In practice, analysis tools generally require some user input. For example, users of the AbsInt aiT tool specify the starting and stopping points of the analysis and some aspects of the microprocessor's configuration [6]. When the analysis cannot determine details such as loop bounds, users supply these also. A tool given faulty data might produce faulty output. As a result, developers must demonstrate the adequacy of all tool inputs to justify confidence in the resulting WCET bound.

Modern processors are becoming more complex. For example, they now employ branch prediction, speculative execution, multi-layer caches, and sometimes even multiple cores. This complexity makes static WCET analysis difficult [7]. Static WCET analysis tools are based on abstract models of processor timing. If these models are incorrect, analysis results might be incorrect. As a result, developers must demonstrate that the tools they use are fit for purpose. In practice, developers of some tools can provide tool qualification evidence as required by standards such as RTCA DO-178B and ISO 26262 [2, 6, 8]. However, like software safety evidence, tool qualification evidence is never perfect: some possibility of error, however small, remains.

No current tool models all of timing-related internals of today's most complex embedded microprocessors. As a result, static WCET tools are not available for some embedded microprocessors (e.g. the Freescale MPC P4808 [9]). Moreover, it is usually more work to extend a tool to a new processor than to devise a means of measuring execution time on that processor [5].

For some processors, static timing analysis tools make conservative simplifying assumptions in order to achieve usefully quick analysis. For example, consider instruction caches and their timing effects. Instructions at the beginning of each basic block might or might not be in the cache depending upon the path taken to reach the block. One classic analysis approach uses a call graph to determine whether all, some, or no possible prior basic blocks leave a given instruction in the cache [10]. Unless all possible paths would yield a hit, the analysis assumes a miss. While such assumptions are deliberately pessimistic to avoid underestimate, they can and do lead to overestimate. Most modern static WCET analysis tools that model instruction cache use some variation on this technique and make some variation on this pessimistic simplifying assumption [5].

In practice, a static analysis result within 20% of the highest WCET observed in testing is considered very good. This has been achieved for a simple system written in SPARK and running on a 68020 microprocessor [11]. More recent work using state of the art techniques claims similar figures [12, 13]. Good results can sometimes be obtained when developers expend substantial effort to optimise the analysis result, e.g. by accurately modelling the software's control flow. However, complex features such as multi-level caches can result in overestimates exceeding 100% in some cases [14]. Static analysis reports usefully tight bounds for some software on some processors, but it cannot do so for all software on all processors.

2.1.2. Dynamic Approaches to Determining WCET

Dynamic WCET analysis techniques can be split into three categories: *high water mark*, *probabilistic*, and

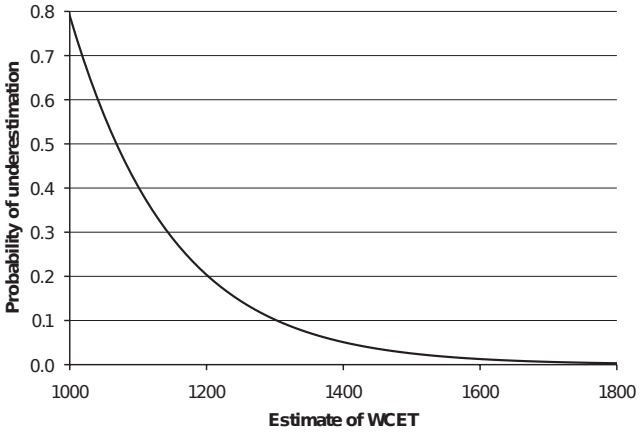

FIGURE 1. Example WCET distribution (data taken from [15]).

search-based.

High water mark. In high water mark approaches, analysts use the longest execution time observed in testing as a WCET estimate. Unfortunately, it is not generally possible to determine either the likelihood or degree of underestimate.

Probabilistic. In probabilistic approaches, analysts use Extremal Value Theory (EVT) to fit observed execution times to a distribution such as the example presented in Figure 1 [15, 16]. By selecting a WCET estimate from further to the right of the distribution, a developer decreases the likelihood of underestimation. However, high confidence is expensive. Moreover, analysts using these approaches typically do not question the validity of the data used to form the distribution, making the result uncertain [17].

Search-based. Search-based techniques have been shown to be relatively successful [18]. Given appropriate objectives, they can solve relatively complex problems [19, 20]. However, they are optimistic. Moreover, their accuracy is subject to randomness.

In all cases, as with static analysis, developers must consider the correctness of any user-supplied data. Developers must also show that any tools used are fit for purpose. Moreover, if instrumentation is used to obtain timing measurements, the process of instrumenting a test binary might alter its timing characteristics. Developers must show that their test data accurately reflects the runtime of the system as it will operate in the field.

2.1.3. Hybrid Approaches to Determining WCET
Hybrid approaches combine static analysis and measurement [16, 21, 22]. In this approach, the analysis tool divides the software into *blocks*. The analyst then executes the software, using the tool to measure the execution time(s) of each block. Finally, the tool combines the measurement values to produce a WCET estimate as in the static analysis approach. In its simplest form, the analysis uses the highest observed execution time for each block. Researchers have proposed more complicated analyses that take inter-block dependencies into account [23].

Intuition suggests that, for a given test suite, hybrid approaches are less likely to underestimate WCET than high water mark approaches. However, both underestimation and overestimation remain possible. The likelihood of underestimate is unknown. The likelihood and degree of overestimation vary with the complexity of the hardware architecture and may, in some cases, be large [24].

As in static and dynamic approaches, user-supplied inputs and tool correctness are a concern. Moreover, as in dynamic approaches, developers must show that the timing test data accurately reflects the delivered system.

2.2. WCRT Analysis Relies Upon Simplifying Assumptions

Developers have used WCRT analysis as safety evidence in critical systems such as avionics [25]. Typical WCRT analysis approaches use either static scheduling [26], fixed priority scheduling [27], or a mix of the two [28]. Researchers have proved that these approaches are mathematically exact so long as certain assumptions hold. These assumptions include:

- There are no overheads, e.g. no interrupts and no cache-related preemption delay
- There are no dependencies, i.e. the runtime of each task is independent of the execution of other tasks
- The WCET figures used are the exact WCETs of the tasks *in the context of the delivered system*

The issue of dependencies and overheads have been addressed in the literature [25, 29]. The WCET figures and their surrounding assumptions represent the biggest problem. Even if developers used a technique that produced exact values, WCET estimation is typically carried out under conditions that differ from the runtime environment of the delivered system. For example, WCET estimation is frequently carried out under the simplifying assumption that each task will run atomically as the sole task on the processor.

Where simplifying assumptions are false, developers compensate for their effect on WCRT analysis results. Researchers have developed techniques that compensate for cache-related timing effects from preemption and caches shared between cores [30, 31, 32]. However, even where techniques for compensating for false assumptions are available, the challenges faced are similar to those already discussed. Developers must show that each such assumption has been adequately

compensated for, e.g., that the effect of interrupts is accounted for by a sufficiently large overhead figure.

2.3. Timing Analysis and Health Monitoring in Practice

In practice, most developers use the high water mark method to estimate WCET [5]. During WCRT analysis, developers sometimes add both an engineering safety margin and limited adjustments for real world effects such as overheads [28, 33]. They assume that it is unlikely that the worst cases for all tasks will occur together and thus unlikely than an overrun in a single task will cause the system to violate a timing requirement. End-to-end timing tests partially confirm this assumption, albeit with limited confidence. While this practice supports *to a degree* the claim that a hard real-time system's timing requirements will be met, it cannot do so to a degree that is adequate when the consequence of a failure is catastrophic.

Developers employ *health monitoring* to mitigate any overruns that occur even though timing analysis shows that they are unlikely. That is, they construct systems so as to detect overruns, insulate other subsystems from the effects of overruns, and trigger recovery mechanisms as needed. For example, a system might respond to the overrun of a control calculation by re-issuing the last control outputs. Alternatively or in combination, developers might use a hardware watchdog timer to reset the microprocessor if deadlines are breached.

Timing analysis and health monitoring are related in two ways. First, it is necessary to know the details of both in order to judge whether the developers have adequately managed a system's timing-related hazards. Second, the appropriateness of health monitoring responses depends upon system timing characteristics that are assessed through timing analysis. For example, suppose that timing analysis establishes that overruns are vanishingly unlikely. In that case, it is reasonable to assume that a deadline miss could only result from a hardware failure that derailed the computation. As a result, it would be reasonable to use a hardware watchdog to reset the microprocessor. Suppose instead that developers used a high water mark approach, that the test suite was not large enough, and that they did not apply a large safety margin. In that case, while rare, overruns must be expected. Resetting the microprocessor in response to these might only aggravate the problem.

2.4. Living With an Imperfect State of the Art

A perceived benefit of many timing analysis approaches is their mathematical rigour [34]. However, it is not generally possible to use state of the art techniques to *prove* that a hard real-time system will *always* meet its timing requirements. For the reasons discussed in Section 2.1, it is not always possible to guarantee that WCET figures are not underestimated. Moreover, both WCET determination and WCRT analysis are complex and must be conducted carefully if the results are to inspire confidence. Health monitoring can be used to mitigate overruns. However, the appropriateness of a given health monitoring approach depends upon the timing characteristics of the system and the hazards to be mitigated.

To judge whether a given system adequately manages its timing-related hazards, developers and assessors must know three things:

1. *The quality of the WCET figure.* This depends on factors such as:
   (a) The likelihood of underestimate
   (b) Whether tools are fit for use
   (c) The quality of inputs such as loop bounds
2. *The quality of the WCRT analysis.* This depends on factors such as:
   (a) The assumptions the analysis is based on
   (b) Compensation for simplifying assumptions
   (c) The soundness of the analysis method
   (d) Whether tools are fit for use
3. *How timing analysis relates to health monitoring.* Determining the appropriateness of health monitoring requires knowing the timing characteristics of the system. Moreover, judging the adequacy of a system requires knowing the details of both timing analysis and health monitoring.

Our thesis is that it is possible to construct a software safety argument that better conveys the crucial timing-related information that is inadequately conveyed by current standards-based and argument-based approaches. In Section 3, we will show how current standards-based and argument-based safety assurance practices inadequately communicate this critical information. In Section 4, we present an example software safety argument that more clearly does so.

3. THE STATE OF THE PRACTICE OF SAFETY ASSURANCE

In some domains, applicable standards require developers to meet specified timing-related development objectives or to follow a prescribed process [2]. In others, developers must address timing as part of a comprehensive *safety case*, i.e. a "structured argument, supported by a body of evidence, that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given environment" [35]. In neither case is the state of the art satisfactory.

3.1. The RTCA DO-178B and DO-178C Standards

In the United States and Europe, software operating on civil air transports is generally developed in conformance with RTCA DO-178B, it successor RTCA DO-178C, or their European equivalents ED-12B and ED-12C [2, 36, 37, 38]. In the process that these standards prescribe, developers capture high-level timing requirements and decompose these across the design into low-level requirements. Developers then demonstrate the satisfaction of these requirements via a combination of test, review, and analysis evidence. DO-178B and its successor require two items of timing-related evidence:

- Paragraph 6.3.4f requires "reviews and analyses" of the source code to (amongst other objectives) "determine ... worst-case execution timing"
- Paragraph 6.4.3a requires "requirements-based hardware/ software integration testing" that could detect "failure to satisfy execution time requirements"

The standard does not explicitly call for the use of a watchdog timer, specify how execution should be scheduled, demand a specific approach to establishing WCET, or constrain how the system should respond to overruns. However, paragraph 6.3.3f does require review and/or analysis of the "partitioning integrity" of the software architecture. This, arguably, forces developers to examine the adequacy of the mechanisms relied upon to implement temporal partitioning.

WCET Figure Quality Unclear. Despite the language of DO-178B, it is not generally possible to precisely 'establish' the WCET of tasks running on modern processors (for the reasons discussed in Section 2.1). RTCA DO-248B, the clarification of DO-178B, explicitly allows developers to use either static or dynamic approaches [39]. Static approaches must account for "all compiler and processor behavior and its impact" on timing. Dynamic approaches are permitted only "if it can be demonstrated that the test provides worst-case execution time". Personal conversation with industrial practitioners, safety assessors, and tool vendors suggests that dynamic approaches are used, and indeed might be more common. The standards offer no guidance on how to demonstrate that these provide WCET and it is not clear what evidence assessors demand in practice.

WCRT Analysis Quality Unclear. DO-178B does not explicitly constrain how developers use task WCET figures to show that timing requirements have been met. Arguably, the standard requires this. However, with no explicit constraint on how this is done, it is not clear which techniques and simplifying assumptions are permissible.

No Connection Between Timing Analysis and Health Monitoring. The standard makes no connection between timing analysis methods and health monitoring. It thus offers developers and assessors no guidance on which combinations are appropriate.

3.2. The ISO 26262 Standard

ISO 26262 is an international safety standard for road vehicles. Part 6 of this standard provides guidance for "product development at the software level" [3]. Software in road vehicles frequently must meet timing-related safety requirements. Accordingly, ISO 26262 includes six time-related requirements:

- The software safety requirements should include any necessary timing constraints (§6.4.2).
- The software architecture should describe temporal constraints on the software components, including tasks, time slices, and interrupts (§7.4.5b).
- If partitioning is used, no software partition may affect the performance, rate, latency, jitter, or duration of other partitions' access to shared resources (§7.4.11a).
- The developers must perform a safety analysis of the architecture and, if necessary, include health monitoring in the system design (§7.4.13–14).
- When developing the architecture, developers must make "an upper estimation of required resources for the embedded software", including "the execution time" (§7.4.17).
- Software testing must include resource usage tests to confirm that the execution time allocated to each task is sufficient (§9.4.3, §10.4.3).

In addition, the standard requires developers to verify "software unit design and implementation", using a variety of static techniques, including "inspection", "semi-formal verification", control and data flow analysis, "static source code analysis", and "semantic code analysis" [8]. The text of this requirement does not explicitly mention timing properties. Moreover, it only recommends (rather than "highly" recommends) semantic code analysis, even for the most critical software. As a result, we do not interpret ISO 26262 as requiring static analysis of WCET (although we would recommend such analysis where practicable).

WCET Figure Quality Unclear. ISO 26262 does not specify how the resource usage tests are to be conducted. If developers statically analyse WCET, the standard does not specify how inputs are to be verified. The quality of a WCET figure established or confirmed by testing varies with the technique used, the size of the test set, and the selection of test cases. ISO 26262 offers no guidance on what form of timing tests is appropriate.

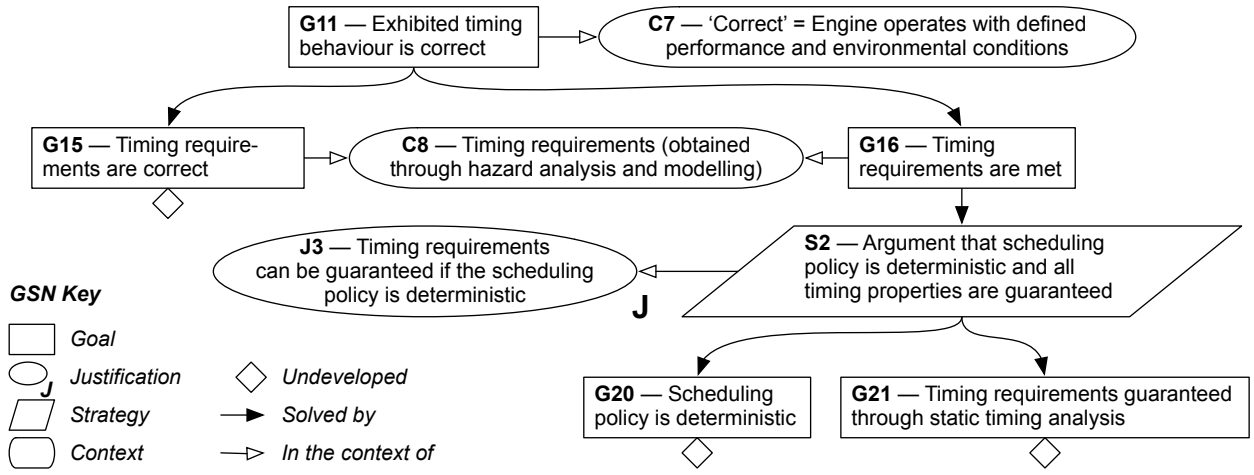

FIGURE 2. Typical timing argument (taken from [40]).

WCRT Analysis Quality Unclear. ISO 26262 requires developers to specify temporal constraints on components. However, it does not specify a process for WCRT analysis based on those WCET figures. While such analyses typically make use of simplifying assumptions, the standard offers no guidance on what is appropriate.

No Connection Between Timing Analysis and Health Monitoring. ISO 26262 explicitly recommends monitoring of execution time to detect overruns. However, it makes no connection between the quality of timing analysis and how overruns are handled.

3.3. Defence Standard 00-56

UK Defence Standard 00-56 takes a different approach to system safety than ISO 26262. Rather than prescribing elements of the development process, Def Stan 00-56 requires developers to create and maintain a safety case [35].

When a system contains software, its safety argument might depend, in part, on the software's behaviour. For example, a fly-by-wire aircraft design creates a hazard: an aircraft might become uncontrollable if software outputs are not computed on time. To argue that this hazard has been adequately managed, developers might cite evidence arising from WCET and WCRT analyses.

The main text of Def Stan 00-56 offers no guidance for software development or for software-related portions of the safety case. However, part 2 of the standard offers guidance on compliance with the standard's main text [41]. Part 2 gives the following advice on the use of analysis evidence for software components but also highlights that run-time errors, which could be due to inaccurate analysis, have to be accounted for:

> Analysis may be used to provide evidence that the safety requirements are satisfied and the derived safety requirements of the complex electronic element hold. Such analyses may include timing (e.g. worst case execution times), use of resources, computational accuracy, possibility of run-time error and functional properties.

The standard requires the developers' safety argument to be "compelling, comprehensible, and valid" [35]. Arguably, a safety argument would not meet these criteria if its treatment of software did not demonstrate the combined adequacy of any timing analysis and health monitoring mechanisms. However, current safety arguments do not do this well.

3.4. Typical Timing Arguments

Figure 2 presents a typical approach to arguing about timing. The argument concerns an engine control computer and is presented in the graphical Goal Structuring Notation (GSN) [42, 43, 44]. The arguer contends that the system will exhibit correct timing behaviour because a correct set of timing requirements (goal G15 and context C8) are met (goal G16). Since a deterministic scheduling policy is used (goal G20), static timing analysis (goal G21) is sufficient to show that the timing requirements have been met. While this was not shown in the original argument, goal G20 might be solved by reference to the scheduling policy used. In a typical argument, goal G21 would be solved by citing the WCET estimation and WCRT analysis as evidence.

This argument was written in 1996 [40]. Nevertheless, its logic reflects current timing analysis practice including that found in academic papers [5, 33], recommended in standard textbooks [26, 45, 46, 47], and reported in industrial papers [34, 48]. This is the logic that underpins the standards we have discussed. Reliance upon timing analysis is typical, although in other arguments this might be backed up by evidence showing that the system has behaved as expected in testing or in practice. The assumption codified in justification J3 is also typical.

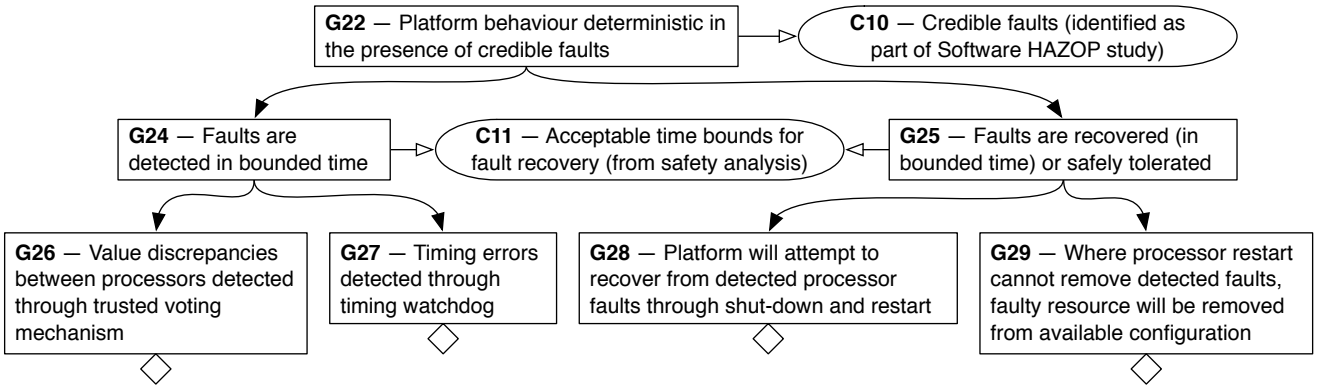

FIGURE 3. Typical health monitoring argument (taken from [40]).

WCET Figure Quality Unclear. In typical safety arguments, the WCET estimate would be cited as evidence with no argument about its quality. *If* the reader of a safety case report is aware that WCET analysis quality might be an issue, he or she could read the analysis report, judge the analysis quality, and compare that judgement with his or her understanding of the importance of WCET analysis evidence to the argument as a whole. However, the argument structure does not make clear to the reader that analysis quality is an issue requiring attention.

WCRT Analysis Quality Unclear. In typical safety arguments, the WCRT analysis would be cited with no argument about the analysis process, its assumptions, or the quality of its result. Again, *if* a reader was aware that simplifying assumptions might be an issue, he or she could read the analysis report and make a judgement. However, once again, the argument structure does not draw the reader's attention to this potential issue.

No Connection Between Timing Analysis and Health Monitoring. Figure 3 presents a health monitoring argument taken from the same safety case as Figure 2. The arguer claims that timing overruns will be detected through the use of a reliable watchdog timer (goal G27) and handled by restarting the processor (goal G28). However, the argument does not explicitly connect the issues of timing analysis and health monitoring, or deal with the issue of duration and frequency of failure. The latter of these is particularly important as there is plenty of evidence that unreliable systems lead to distrust of the systems and consequently accidents [49, 50]. In addition, a slow response to failures can lead to hazards. Finally the more time the system spends in a failed state, the less useful it is. Therefore the time taken to recovery and the frequency of failures should be reasoned about.

3.5. State of the Art Modular Software Safety Arguments

The Industrial Avionics Working Group (IAWG) aims to produce a state of the art modular software safety case process. While the IAWG effort continues, early results have included an approach to arguing about software timing [51]. In this approach, the claim that "all critical operations complete within the allotted time" is broken down into five sub-claims:

1. *"All users of throughput have been adequately identified."* That is, all scheduled applications, interrupts, scheduling overheads, and other overheads (e.g. DMA, cache) have been identified.
2. *"Each user of throughput has an execution time budget."* This budget is identified in the system design documentation.
3. *"All users of throughput stay within individual execution time budgets."* This claim is supported by: (a) analysis that identifies the worst case; (b) a timing measurement of the worst case; and (c) temporal partitioning.
4. *"Cumulative use of throughput is within overall execution time budget."* This claim is supported by: (a) analysis of the scheduling algorithm showing that the "total of [the] throughput budgets is schedulable"; (b) end-to-end tests showing that the "test cases meet [their] timing constraints"; and (c) (optionally) unspecified evidence showing that "overruns do not affect critical operations."
5. *"Scheduling software implements the scheduling algorithm."* This would be supported by testing and analysis evidence.

WCET Figure Quality Unclear. The IAWG modular software safety argument claims that the worst case has been identified. This is implausible. Identifying the worst case cannot be done by testing without perfect coverage, which is not generally attainable. It is also

impossible to identify the worst case using an analysis tool that does not exactly establish ET in all cases: any degree of overestimate or underestimate might cause the tool to evaluate a shorter-running case as worse than a longer-running case. The claim to have identified the worst case cannot be substantiated beyond any reasonable doubt. However, the claim does not specify the level of confidence that is actually achieved. As a result, the quality of WCET figures cited in the argument is questionable.

WCRT Analysis Quality Unclear. The IAWG argument addresses some issues better than typical safety arguments do. In particular, it cites analysis that identifies all overheads. However, the impact of simplifying assumptions on the analysis result remains unclear.

No Connection Between Timing Analysis and Health Monitoring. The IAWG argument cites evidence of temporal partitioning and, optionally, evidence showing that overruns do not affect critical operations. However, it does not explicitly identify the interplay between these and the cited timing analysis evidence.

4. IMPROVING ARGUMENTS ABOUT SOFTWARE TIMING

The second part of our thesis is that it is possible to construct a software safety argument that better conveys the crucial information that existing approaches obscure. In this section, we present an example of such an argument that illustrates six benefits of a well-structured software safety argument:

1. *WCET Assumptions.* Our argument captures and justifies assumptions about the WCET estimation context (e.g. atomic operation, processor architecture) and data (e.g. distribution of inputs).
2. *WCET Quality.* Our argument states the quality of WCET estimates in quantitative terms.
3. *WCRT Assumptions.* Our argument captures and justifies assumptions about the WCRT analysis context (e.g. overheads, processor architecture) and system inputs (e.g. accuracy of WCET).
4. *WCRT Quality.* Our argument clearly states the confidence inspired by the WCRT analysis analysis.
5. *Health Monitoring.* Our argument explains how health monitoring justifies confidence that a system meets its safety requirements despite the limitations of timing analysis and WCET figures.
6. *Traceability to Evidence.* Our argument traces each system hazard through software safety requirements to evidence. It explains what each form of evidence (e.g. task timing tests, end-to-end timing tests) shows and how much confidence each form should inspire.

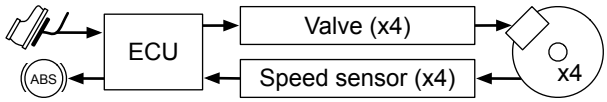

FIGURE 4. Computer assisted braking system.

Our example argument shows one approach to WCET and timing analysis. However, our choice of specimen is not a recommendation of that approach over the others discussed in Section 2. Each approach has distinct advantages, disadvantages, and issues of confidence. Each requires a different argument. We leave specific argument patterns for each common approach for future work.

4.1. Specimen Application: Computer Assisted Braking

Our example argument concerns a specimen Computer Assisted Braking (CAB) system. We chose this system both because relatively comprehensive safety cases for it are in the public domain [52, 53] and because CAB is representative of systems in a number of domains, e.g. Anti-Lock Braking (ABS) systems for cars, and Anti-Skid systems for aircraft and trains.

4.1.1. CAB System Architecture

Figure 4 illustrates the CAB system that is the subject of our case study. The system comprises:

1. An Electronic Control Unit (ECU)
2. A brake pedal sensor
3. Four wheel speed sensors (one per wheel)
4. Four valves, each of which can relieve or restore braking force at one wheel
5. A dashboard indicator light to warn the user when the system is not available

The braking system is primarily hydro-mechanical, not brake-by-wire. The CAB components are not replicated. The ECU implements software health monitoring that can detect task overruns and take fine-grained remedial action. For example, if control calculations are not completed on time, the software re-issues its last outputs to the valves. The ECU also implements simple hardware health monitoring that resets the microprocessor when a hardware watchdog timer expires. Software on the ECU is scheduled using fixed priority scheduling [25].

Our specimen CAB system is simple but realistic. ABS has been implemented in cars using similar arrangements, although current systems might also feature brake-by-wire and use a separate microcontroller for each wheel. This system is an appropriate choice

for our example because, despite its simplicity, arguing about its timing properties requires the same kind of arguments that a system with more features or more processors might.

4.1.2. CAB System Hazards

Wilson et al. have conducted a hazard analysis on a similar system [52]. Several of the hazards that they identify will also be present in our system. For the purpose of this paper, we focus on one such hazard:

Hazard ID:	Lock Up
Description:	One to four of the wheels stops turning while the vehicle is in motion
Severity:	Catastrophic

4.2. Example Argument For The CAB System

Figure 5 shows the top level of a safety argument for the CAB system. Goal GSafety represents the main claim, that the system is acceptably safe to operate. Context elements CBrakeSys, CSafety, and COpContext provide operational definitions of 'Braking System', 'acceptably safe', and 'operating context', respectively.

As is typical and recommended in safety arguments, we support this claim in part by arguing that all identified hazards are acceptably managed [54]. We support the claim that the Lock Up hazard is acceptably managed by showing that contributions from the software and other subsystems are acceptably managed. In Figure 5, we focus on one particular software contribution to this hazard: the possibility that the software will be late in commanding the valves to mitigate Lock Up.

A different application would have a different operating context, different hazards, and different safety requirements. Nevertheless, this example illustrates two improvements that could be used elsewhere:

1. It demonstrates traceability from hazards to safety requirements (and, in subsequent figures, to evidence).
2. It uses multiple deadlines at different levels of confidence to model the impact of health monitoring.

As developers choose means of managing each identified hazard, they record safety requirements that describe successful management. In Figure 5, the goals GSSR10, GSSR11, and GSSR12 represent three tiers of software management of the Late Correction contribution to Lock Up. In general, we require that the Anti-Lock function activate within 50 ms of the onset of wheel lock up (goal GSSR10). However, for the reasons given in Section 2, we could not satisfy this requirement with sufficient confidence. Goals GSSR11 and GSSR12 represent requirements to meet relaxed deadlines with the greater confidence made possible by health monitoring strategies.

We represent confidence using categories analogous to the Safety Integrity Levels (SILs) defined in some safety standards. In our scheme, confidence level A is the highest. One could, instead, represent confidence in terms of the likelihood (over some assumed distribution of inputs) that the deadline would be missed in operation. However, unless the required confidence was low enough that practical system-level testing could provide it, it might be difficult to argue convincingly that such a requirement had been met. For example, we do not know how to quantify the effects of human error and other sources of epistemic uncertainty.

This portion of the example argument links hazards to safety requirements and the confidence that must be inspired by health monitoring and timing analysis. In the following subsections, we illustrate how to complete the link to evidence and the confidence that evidence inspires.

4.2.1. Arguing About Confidence In WCET Figures

Figure 6 illustrates part of an argument module focused on WCET claims for our specimen system. A different approach to determining WCET would require a different argument. Nevertheless, this example illustrates two improvements that could be brought to other arguments:

1. Context element CWCET.C clearly communicates the confidence level associated with WCET figures.
2. Context elements CWCET.NI, CSysHW, and CSysOS summarise the contextual assumptions that underpin the analysis.

Communicating Confidence. The confidence figure in CWCET.C is part of the WCET argument module's contract. That is, this argument will defend the claim to have achieved that level of confidence so that other portions of the argument can rely upon the truth of that claim. As elsewhere in the argument, we might have stated the confidence in terms of a SIL rather than a failure rate.

Confidence in the WCET figures is limited by epistemic and aleatoric uncertainty from several sources:

1. *Configuration Management.* Analysis of the wrong binary might produce the wrong WCET figure. Goal GBPCM.WCET (supported in another module, not shown) show that we have taken adequate care.
2. *Tool Fitness.* An error in the hybrid WCET analysis tool might also produce an incorrect WCET figure. We cite tool qualification evidence to show that the tool is fit for use in this application.
3. *Tool Inputs.* Erroneous user-provided loop bounds or other tool inputs might also result in incorrect output. Evidence for goals such as GWCETAI.LB shows that each input is of adequate quality.
4. *Test Coverage.* Timing tests that achieve incomplete coverage might miss the worst case. Goal GWCETAI.-BBT represents the claim that testing-derived basic block timing data is sufficient to meet our quality goals.

We argue that test-derived basic block timing data is sufficient because our tests achieved suitable code coverage. Context element ABBTInput represents our definition of 'suitable'. Since our chosen timing tool statically infers loop bounds, this definition does not include loop bounds (or recursion depth). (I.e., basic blocks do not include loops). Ongoing research is addressing the relationship between confidence and timing test coverage [17]. Nevertheless, expert developers and safety assessors routinely use expert judgment to make assumptions about what coverage is appropriate. Such assumptions should be documented.

Communicating Assumptions. CWCET.NI, CSysHW, and CSysOS summarise assumptions that form part of the

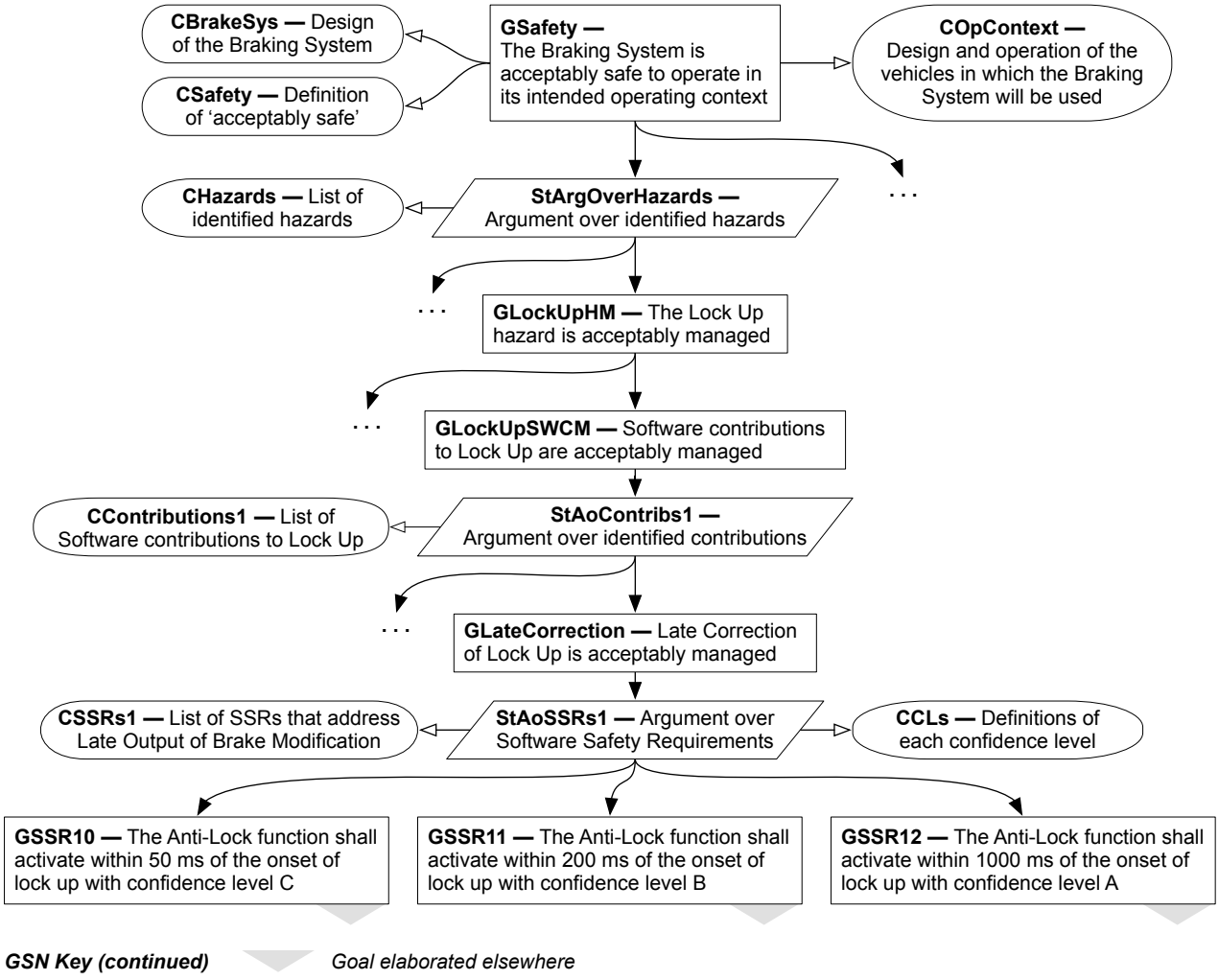

FIGURE 5. Top level of CAB system safety argument.

argument module's contract. As we show in Section 4.2.2, interlinked portions of the argument must demonstrate compatibility with these assumptions.

Other Cases. Most forms of WCET analysis use tools. Thus, the basic strategy of arguing over a tool and its inputs should be generally useful. However, suppose we had used a dynamic approach as discussed in Section 2.1.2. The tool might be simpler and use fewer inputs (e.g. eliminating GWCETAI.LB and GWCETAI.CPUC), but we would still require evidence of its fitness. The coverage criteria would differ depending on the exact approach taken (e.g. statistical versus high water mark).

Had we instead used a purely static approach, we would not need the sub-argument headed by GWCETAI.BBT at all. However, much of the remaining argument would be the same as for the example hybrid approach. For example, we would still need GWCETTC: static analysis tools are complex, and some have been based on incorrect information provided by microprocessor vendors [55].

Regardless of the approach, details of the argument will depend on details of the approach. For example, some tools restrict developers to a subset of a programming language. Users of RapiTime and C should not use `goto` because paths containing `goto` are ignored in WCET analysis [56]. Some language constructs promote repeatability, which is important in dynamic approaches [57]. The coding guidelines mentioned in goal GWCETAI.CG must be appropriate given the approach and confidence targets.

4.2.2. Arguing About Confidence In Timing Analysis Figure 9 illustrates part of an argument module focused on timing analysis. While this argument reflects a schedulability analysis, there are other approaches to timing analysis. Nevertheless, it illustrates three improvements that could be brought to other arguments:

1. Context element CTiming.HC clearly communicates our confidence in the timing analysis.

2. Context elements summarise the contextual assumptions underpinning the analysis.

3. The argument demonstrates that the assumptions underpinning the timing analysis are compatible with those underpinning the WCET figures.

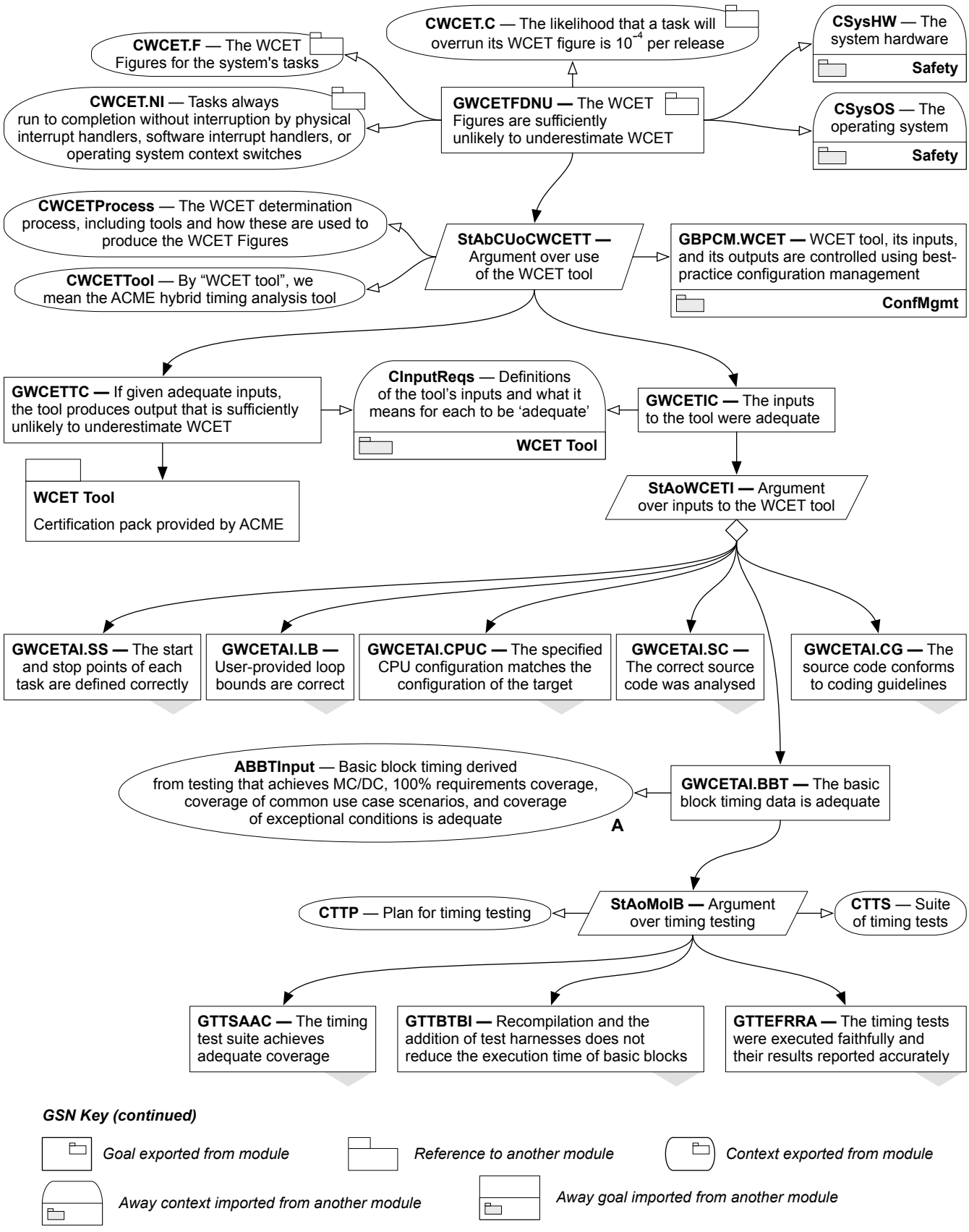

FIGURE 6. Argument over WCET estimation in the CAB system.

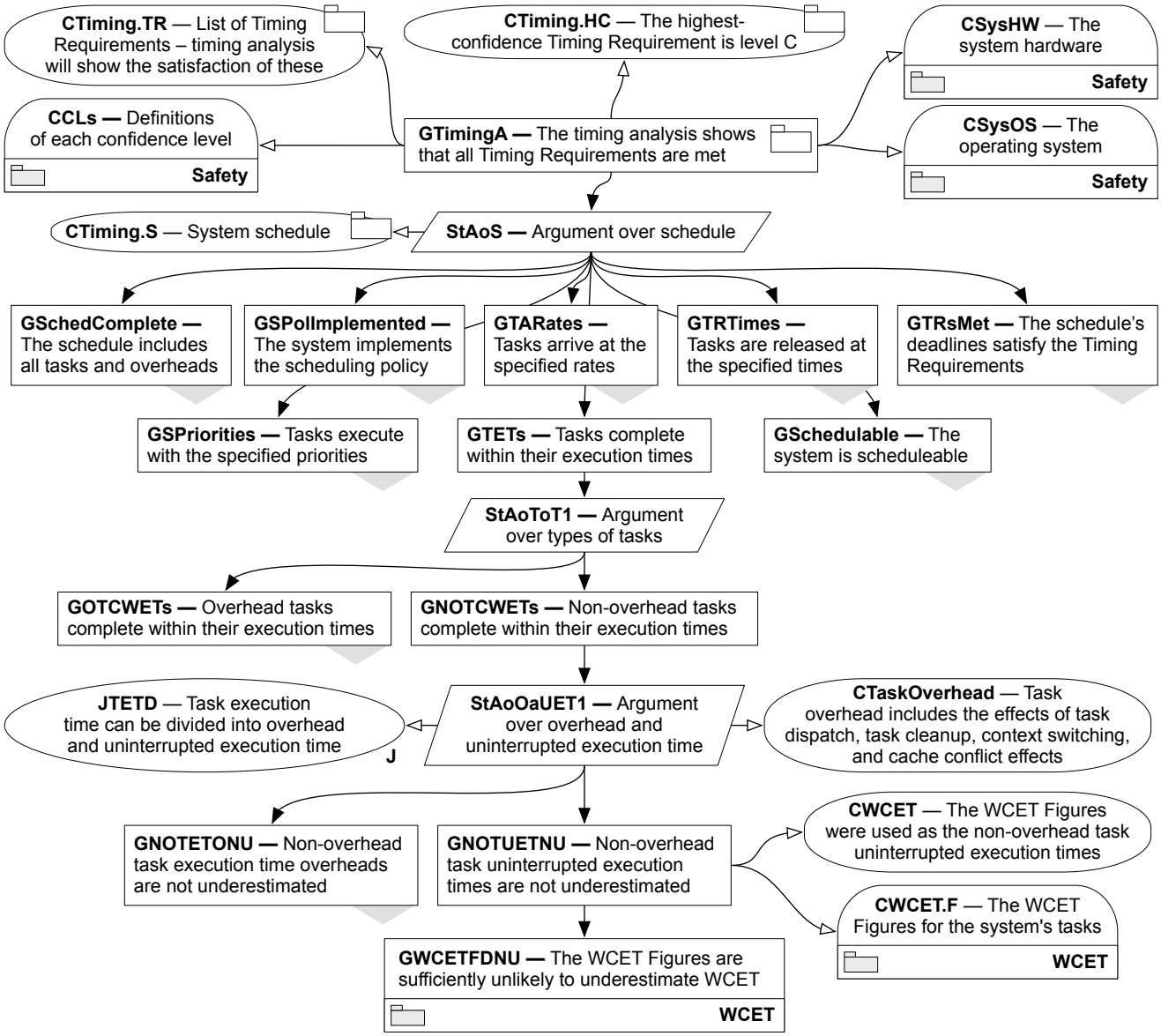

FIGURE 7. Argument over timing analysis in the CAB system.

Compatibility of Assumptions. The argument supporting goal GNOTCWETs illustrates how the timing analysis process compensates for the assumptions made in the WCET and timing argument module. Because the WCET measurement process assumed that each task ran in isolation, we divide each timing figure into uninterrupted execution times and overheads. We use the WCET argument module to support a claim that the uninterrupted execution times are not underestimated (goal GNOTUETNU). We must separately argue that the overhead figures used are sufficient to convert uninterrupted execution times into figures that do not underestimate runtimes in the real system context (goal GNOTETONU, support not shown).

Table 8 illustrates how the *argument contract* between the WCET and timing analysis demonstrates the compatibility between their separate assumptions [58, 44]. Argument contracts expresses the relationship between argument modules and include elements not shown in the figure. In this case, the relationship is that the WCET module's goal GWCETFDNU supports the timing module's goal GNOTUETNU. This support requires compatibility between the modules' context. In this case, we state why we believe that it is acceptable to assume that tasks run in isolation for the purpose of using WCET figures in the timing analysis.

Other Cases. Suppose that the target was a multicore microprocessor. Developers could either analyse WCET in the context of what might be running on the other cores or assume during WCET analysis that each task has exclusive access to all shared caches, memories, and other devices. Either choice presents assurance difficulties that are revealed by the argument structure. Researchers have proposed multicore architectures that make the time cost of using shared resources either predictable or truly

Goals matched between participant modules			
Goal	**Required by**	**Addressed by**	**Goal**
GNOTUETNU	Timing Analysis	WCET	GWCETFDNU

Collective context of participant modules held to be constant	
Context	**Rationale**
CWCET.NI	While tasks do not always run to completion without interruption, goal GNOTUETNU is concerned with 'uninterrupted execution times'. WCET figures estimated under the assumptions listed in CWCET.NI will not underestimate uninterrupted execution times.

FIGURE 8. Extract from contract between the WCET and timing analysis modules

random [59, 60, 61]. While such architectures would make the assumption easier to compensate for, implementations are not yet commercially available. Researchers have also proposed static analysis techniques for modelling shared caches [62, 63, 64]. While these would obviate the need for the assumption, there are no commercially available static WCET analysers that produce usefully tight execution time bounds for platforms where cores share parts of a memory hierarchy. Unless developers statically schedule all access to shared components, we know of no way to argue high confidence in usefully-tight WCET figures for software running on most modern multicore platforms.

4.2.3. Linking Health Monitoring To Timing Analysis Figure 9 illustrates part of our argument about software contributions to CAB system hazards. Other systems' hazards will differ. Nevertheless, the figure illustrates two improvements that could be brought to other arguments:

1. Health monitoring supplements timing analysis to show that timing requirements will be met.

2. End-to-end timing test evidence and historical evidence increase confidence still further.

Arguing About Health Monitoring. Goal GSSR11 represents the claim that the Anti-Lock function activates within 200 ms of the onset of lock up with confidence level B. We argue that this claim is true because the vast majority of the time, each software task makes progress (goal GSCFGPR2), and when it does not, the system restarts the task (goals GSWDEDR and GRestart). The assumption behind this logic, ASoftErrors, is that restarting a task can clear many sources of error that would cause it to miss deadlines.

We must also show that it is acceptable to restart tasks even if the problem is *not* a soft error. The design of our example system is such that, if restarting the task fails to clear the problem, the system will go on to reset the processor or even go on to disable itself and illuminate the dashboard warning light. Safety analysis of the design should provide support for goal GNoHarm (not shown).

Health monitoring affects factors other than response time. For example, health monitoring and recovery affects availability. We would expect a safety requirement for system availability to *also* cite evidence related to health monitoring. In this example, we focus solely on timing.

Multiple Sources of Timing Evidence. Goal GSSR10 is supported by multiple forms of timing evidence. We claim that this increases confidence because the evidence is diverse. We would have less confidence in goal GSSR10 if, for example, the end-to-end timing tests were based on the same set of timing test cases as the timing analysis. Assurance claim point 41 identifies the subject of a *confidence argument* that provides backing for our claim of independence [65]. The confidence argument (not shown) gives our reasons for claiming that these sources of evidence are independent and discusses and justifies any remaining assurance gaps.

5. CONCLUSIONS

Typical approaches to demonstrating that a software system meets its timing requirements rely upon a complicated combination of WCET determination, timing analysis, and the provision of health monitoring. The WCET of software running on modern microprocessors can be difficult to determine, and analyses often make incompatible or false simplifying assumptions that must be compensated for. Unfortunately, existing approaches to certification do not address these complexities well. We have presented example safety arguments to demonstrate how to better communicate the quality of WCET figures and timing analysis and how these are complemented by health monitoring. Our argument illustrates how developers can make six improvements to their method of demonstrating the safety of real-time systems:

1. It documents and justifies the assumptions that underpin WCET analysis
2. It clearly communicates the quality of WCET figures
3. It documents and justifies the assumptions that underpin WCRT analysis
4. It clearly communicates the quality of the WCRT analysis
5. It explains how health monitoring boosts confidence that the system meets its timing requirements
6. It shows how evidence justifies confidence that safety requirements are met and system hazards managed

While our example argument focuses on one specimen system, the way in which it achieves these gains can be transferred to safety arguments for systems using other approaches to determining WCET and analysing timing.

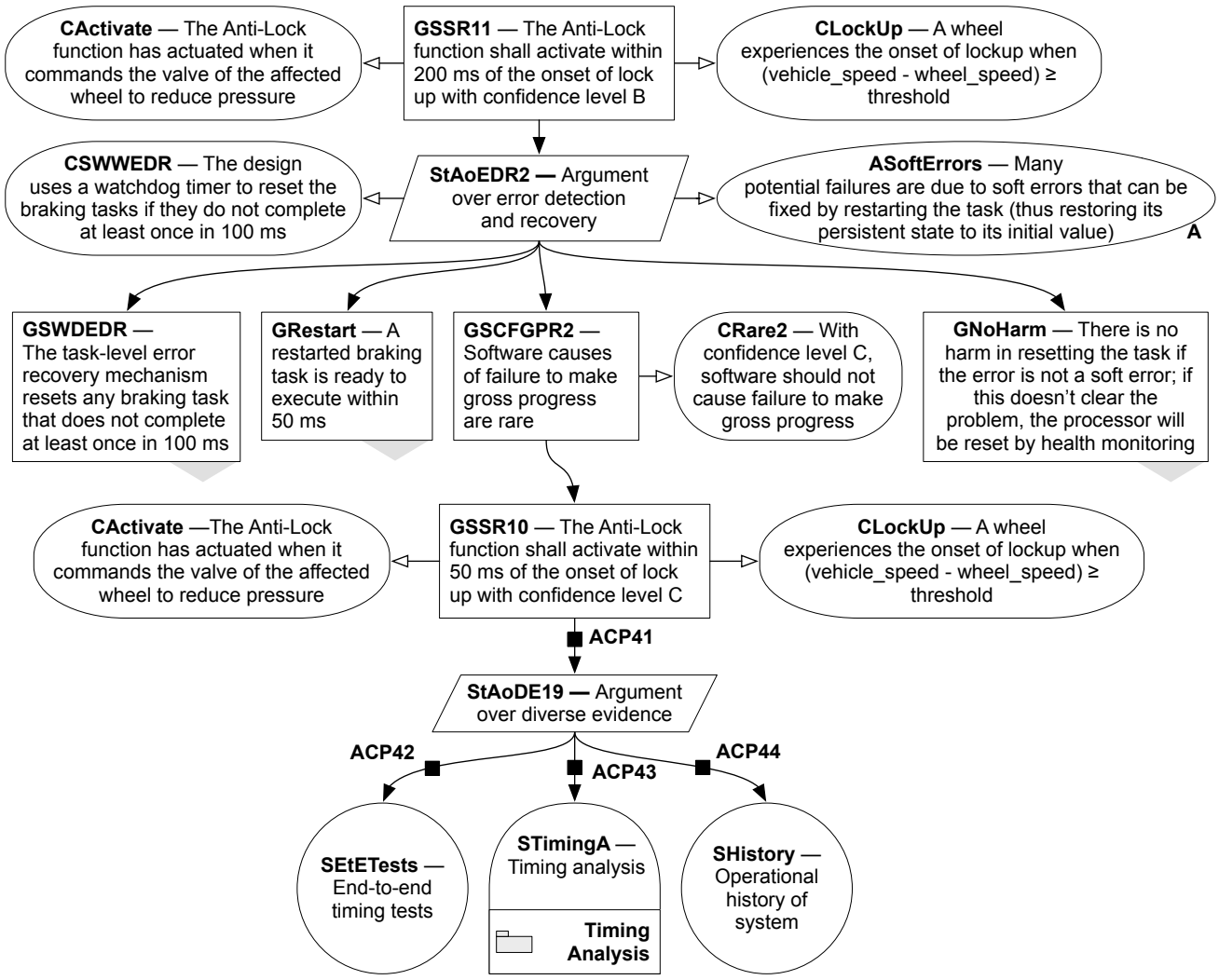

FIGURE 9. Argument over health management in the CAB system.

ACKNOWLEDGMENT

We thank the reviewers for their helpful comments. This work was supported by the Swedish Foundation for Strategic Research (SSF) as part of the SYNOPSIS project.

REFERENCES

[1] Burns, A. and McDermid, J. A. (1994) Real-time safety-critical systems: analysis and synthesis. *Software Engineering Journal*, **9**, 267–281.

[2] RTCA DO-178B (1992) *Software Considerations in Airborne Systems and Equipment Certification*. RTCA, Inc., Washington, DC, USA.

[3] ISO/IEC 26262-6:2011 (2011) *Road vehicles — Functional safety — Part 6: Product development at the software level*. International Organization for Standardization, Geneva, Switzerland.

[4] Park, C. and Shaw, A. C. (1990) Experiments with a program timing tool based on source-level timing schema. *Proc. of the 11th Real-Time Systems Symposium (RTSS)*, Lake Buena Vista, FL, USA, 5–7 December, pp. 72–81. IEEE Computer Society, Washington, DC, USA.

[5] Wilhelm, R. et al. (2008) The worst-case execution-time problem—Overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, **7**, 36:1–36:53.

[6] AbsInt. aiT worst-case execution time analyzers. Web page: `http://www.absint.com/ait/index.htm`.

[7] Bate, I., Conmy, P., and McDermid, J. (2000) Generating evidence for certification of modern processors for use in safety-critical systems. *Proc. of the 5th IEEE International Symposium on High Assurance Systems Engineering (HASE)*, Albuquerque, NM, USA, 15–17 November, pp. 125–134. IEEE Computer Society, Washington, DC, USA.

[8] ISO 26262-8:2011 (2011) *Road vehicles — Functional safety — Part 8: Supporting processes*. International Organization for Standardization, Geneva, Switzerland.

[9] Freescale Semiconductor, Inc. P4080 product summary page. Web page: `http://www.freescale.com/`

webapp/sps/site/prod_summary.jsp?code=P4080.

[10] Arnold, R., Mueller, F., Whalley, D., and Harmon, M. (1994) Bounding worst-case instruction cache performance. *Proc. of the 15th Real-Time Systems Symposium (RTSS)*, San Juan, Puerto Rico, 7–9 December, pp. 172–181. IEEE Computer Society, Washington, DC, USA.

[11] Chapman, R. (1995) Static Timing Analysis and Program Proof. DPhil thesis University of York York, UK.

[12] Ferdinand, C. (1997) Cache Behavior Prediction for Real-Time Systems. PhD thesis Saarland University Saarbrücken, Germany.

[13] Grund, D., Reineke, J., and Gebhard, G. (2011) Branch target buffers: WCET analysis framework and timing predictability. *Journal of Systems Architecture, special edition on Design and Optimization for Embedded Real-Time Computing Systems and Applications*, **57**, 625–637.

[14] Chattopadhyay, S. and Roychoudhury, A. (2009) Unified cache modeling for WCET analysis and layout optimizations. *Proc. of the 30th Real-Time Systems Symposium (RTSS)*, Washington, DC, USA, 1–4 December, pp. 47–56. IEEE Computer Society, Washington, DC, USA.

[15] Edgar, S. and Burns, A. (2001) Statistical analysis of WCET for scheduling. *Proc. of the 22th Real-Time Systems Symposium (RTSS)*, London, UK, 3–6 December, pp. 215–224. IEEE Computer Society, Washington, DC, USA.

[16] Bernat, G., Colin, A., and Petters, S. M. (2002) WCET analysis of probabilistic hard real-time systems. *Proc. of the 23rd Real-Time Systems Symposium (RTSS)*, Austin, Texas, USA, 3–5 December, pp. 279–288. IEEE Computer Society, Washington, DC, USA.

[17] Wheeler, S., Bate, I., and Bartlett, M. (2011) Video subset selection for measurement based worst case execution time analysis. *Proc. of the 6th International Symposium on Industrial Embedded Systems (SIES)*, Västerås, Sweden, 15–17 June, pp. 213–222. IEEE Computer Society, Washington, DC, USA.

[18] Wegener, J., Sthamer, H., Jones, B. F., and Eyres, D. E. (1997) Testing real-time systems using genetic algorithms. *Software Quality Journal*, **6**, 127–135.

[19] Khan, U. and Bate, I. (2009) WCET analysis of modern processors using multi-criteria optimisation. *Proc. of the 1st International Symposium on Search Based Software Engineering (SSBSE)*, Windsor, UK, 13–15 May, pp. 103–112. IEEE Computer Society, Washington, DC, USA.

[20] Bate, I. and Khan, U. (2011) WCET analysis of modern processors using multi-criteria optimisation. *Empirical Software Engineering*, **16**, 5–28.

[21] Betts, A., Merriam, N., and Bernat, G. (2010) Hybrid measurement-based WCET analysis at the source level using object-level traces. *Proc. of the 10th International Workshop on Worst-Case Execution Time Analysis (WCET)*, Brussels, Belgium, 6 July, pp. 54–63. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany.

[22] Rapita Systems (2011) RapiTime explained. Electronic white paper: `http://www.rapitasystems.com/downloads/rapitime_explained_white_paper`.

[23] Bernat, G., Burns, A., and Newby, M. (2005) Probabilistic timing analysis: An approach using copulas. *Journal of Embedded Computing*, **1**, 179–194.

[24] Colin, A. and Petters, S. (2003) Experimental evaluation of code properties for WCET analysis. *Proc. of the 24th Real-Time Systems Symposium (RTSS)*, Cancún, Mexico, 3–5 December, pp. 190–199. IEEE Computer Society, Washington, DC, USA.

[25] Bate, I. and Burns, A. (2003) An integrated approach to scheduling in safety-critical embedded control systems. *Real-Time Systems*, **25**, 5–37.

[26] Kopetz, H. (2011) *Real-Time Systems: Design Principles for Distributed Embedded Applications*, second edition. Springer, New York.

[27] Joseph, M. and Pandya, P. (1986) Finding response times in a real-time system. *The Computer Journal*, **29**, 390–395.

[28] Audsley, N. and Wellings, A. (1996) Analysing APEX applications. *Proc. of the 17th Real-Time Systems Symposium (RTSS)*, Los Alamitos, CA, USA, 4–6 December, pp. 39–44. IEEE Computer Society, Washington, DC, USA.

[29] Bate, I. and Burns, A. (1999) An approach to task attribute assignment for uniprocessor systems. *Proc. of the 11th Euromicro Conference on Real-Time Systems*, York, UK, 9–11 June, pp. 46–53. IEEE Computer Society, Washington, DC, USA.

[30] Lee, C.-G., Hahn, J., Seo, Y.-M., Min, S. L., Ha, R., Hong, S., Park, C. Y., Lee, M., and Kim, C. S. (1997) Enhanced analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *Proc. of the 18th Real-Time Systems Symposium (RTSS)*, San Francisco, CA, USA, 2–5 December, pp. 187–198. IEEE Computer Society, Washington, DC, USA.

[31] Lee, C.-G., Hahn, J., Seo, Y.-M., Min, S. L., Ha, R., Hong, S., Park, C. Y., Lee, M., and Kim, C. S. (1998) Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE Transactions on Computers*, **47**, 700–713.

[32] Chattopadhyay, S., Roychoudhury, A., and Mitra, T. (2010) Modeling shared cache and bus in multi-cores for timing analysis. *Proc. of the 13th International Workshop on Software & Compilers for Embedded Systems (SCOPES)*, St. Goar, Germany, 19–21 June, pp. 6:1–6:10. ACM, New York, NY, USA.

[33] Audsley, N. C., Burns, A., Davis, R. I., Tindell, K. W., and Wellings, A. J. (1995) Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, **8**, 173–198.

[34] Hutchesson, S. and Hayes, N. (1998) Technology transfer and certification issues in safety critical real time systems. *Digest of the IEE Colloquium on Real-Time Systems*, York, UK, 21 April, pp. 2/1–2/4. IET, London, UK.

[35] Defence Standard 00-56 (2007) *Safety Management Requirements for Defence Systems, Issue 4, Part 1: Requirements*. Ministry of Defence, Glasgow, UK.

[36] EUROCAE ED-12B (2012) *Software Considerations in Airborne Systems and Equipment Certification*. EUROCAE, Malakoff, France.

[37] EUROCAE ED-12C (2012) *Software Considerations in Airborne Systems and Equipment Certification*. EUROCAE, Malakoff, France.

[38] RTCA DO-178C (2011) *Software Considerations in Airborne Systems and Equipment Certification*. RTCA, Inc., Washington, DC, USA.

[39] RTCA DO-248B (2001) *Final Report For Clarification Of DO-178B "Software Considerations In Airborne Systems And Equipment Certification"*. RTCA, Inc., Washington, DC, USA.

[40] Kelly, T., Bate, I., McDermid, J., and Burns, A. (1997) Building a preliminary safety case: An example from aerospace. *Proc. of the 1997 Australian Workshop on Industrial Experience with Safety Critical Systems and Software*, Sydney, Australia, 3 October. Australian Computer Society, Inc., Sydney, Australia.

[41] Defence Standard 00-56 (2007) *Safety Management Requirements for Defence Systems, Issue 4, Part 2: Guidance on Establishing a Means of Complying with Part 1*. Ministry of Defence, Glasgow, UK.

[42] Kelly, T. P. (1998) Arguing Safety — A Systematic Approach to Managing Safety Cases. DPhil thesis University of York York, UK.

[43] Bate, I. and Kelly, T. (2003) Architectural considerations in the certification of modular systems. *Reliability Engineering and System Safety*, **81**, 303–324.

[44] Attwood, K. et al. (2011) *GSN Community Standard Version 1*. Origin Consulting Limited, York, UK.

[45] Liu, J. W. S. (2000) *Real-Time Systems*. Prentice Hall, Upper Saddle River, NJ, USA.

[46] Burns, A. and Wellings, A. (2001) *Real-Time Systems and Programming Languages: Ada 95, real-time Java, and real-time POSIX*, 3rd edition. Addison Wesley, London, UK.

[47] Buttazzo, G. C. (2004) *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications* Real-Time Systems Series. Springer-Verlag TELOS, Santa Clara, CA, USA.

[48] Souyris, J. (2004) Industrial experience of abstract interpretation-based static analyzers. *Building the Information Society: Proc. of the IFIP 18th World Computer Congress*, Tolouse, France, 22-27 August, pp. 393–400. Springer, Boston.

[49] Leveson, N. and Turner, C. (1993) An investigation of the Therac-25 accidents. *IEEE Computer*, **26**, 18–41.

[50] Aircraft Accident Report No. 4/90 (EW/C1095) (1990) *Report on the accident to Boeing 737–400 — G-OBME near Kegworth, Leicestershire on 8 January 1989*. Air Accidents Investigation Branch, Department of Transport, Aldershot, UK.

[51] IAWG-MSSC-0401 (2010) *IAWG Modular Software Safety Case Process Guidance on Process for an ASAAC Architecture, Issue 1*. Industrial Avionics Working Group, Rochester, UK.

[52] Wilson, S. P., Kelly, T. P., and McDermid, J. A. (1997) Safety case development: Current practice, future prospects. *Proc. of the 12th Annual CSR Workshop of Software-Based Systems*, Bruges, Belgium, 12–15 September 1995, pp. 135–156. Springer, London, UK.

[53] Grunske, L. (2006) Towards an integration of standard component-based safety evaluation techniques with SaveCCM. *Quality of Software Architectures: Proc. of the 2nd International Conference on Quality of Software Architectures (QoSA)*, Västerås, Sweden, 27–29 June, pp. 199–213. Springer, Berlin / Heidelberg, Germany.

[54] Menon, C., Hawkins, R., and McDermid, J. (2009) Interim standard of best practice on software in the context of DS 00-56 Issue 4, Issue 1. Interim Standard of Best Practice SSEI-BP-000001. Software Systems Engineering Initiative, York, UK.

[55] Schoeberl, M. and Puschner, P. (2009) Is chip-multiprocessing the end of real-time scheduling? *Proc. of the 9th International Workshop on Worst-Case Execution Time Analysis (WCET)*, Dublin, Ireland, 1–3 July, pp. 1–11. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany.

[56] Bonenfant, A. et al. (2010) Coding guidelines for WCET analysis using measurement-based and static analysis techniques. Technical Report IRIT/RR—2010-8—FR. Institut de Recherche en Informatique de Toulouse, Toulouse, France.

[57] ISO/IEC TR 24772:2010(E) (2010) *Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages through language selection and use*, 1.0 edition. International Organization for Standardization, Geneva, Switzerland.

[58] Kelly, T. P. (2001) Concepts and principles of compositional safety cases. Research Report COMSA/2001/1/1. University of York, York, UK.

[59] Pitter, C. (2008) Time-predictable memory arbitration for a Java chip-multiprocessor. *Proc. of the 6th International Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES)*, Santa Clara, CA, USA, 24–26 September, pp. 115–122. ACM, New York, NY, USA.

[60] PROARTIS (2010). Probabilistic analysis takes critical real-time safety certification and verification to new level, says EU project. Electronic document.

[61] Ungerer, T. et al. (2010) Merasa: Multicore execution of hard real-time applications supporting analyzability. *IEEE Micro*, **30**, 66–75.

[62] Yan, J. and Zhang, W. (2008) WCET analysis for multi-core processors with shared L2 instruction caches. *Proc. of the Real-Time and Embedded Technology and Applications Symposium (RTAS)*, St. Louis, MO, USA, 22–24 April, pp. 80–89. IEEE Computer Society, Washington, DC, USA.

[63] Zhang, W. and Yan, J. (2009) Accurately estimating worst-case execution time for multi-core processors with shared direct-mapped instruction caches. *Proc. of the 15th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Beijing, China, 24–26 August, pp. 455–463. IEEE Computer Society, Washington, DC, USA.

[64] Li, Y., Suhendra, V., Liang, Y., Mitra, T., and Roychoudhury, A. (2009) Timing analysis of concurrent programs running on shared cache multi-cores. *Proc. of the 30th Real-Time Systems Symposium (RTSS)*, Washington, DC, USA, 1–4 December, pp. 57–67. IEEE Computer Society, Washington, DC, USA.

[65] Hawkins, R., Kelly, T., Knight, J., and Graydon, P. (2011) A new approach to creating clear safety arguments. *Advances in Systems Safety: Proc. of the 19th Safety-Critical Systems Symposium*, Southampton, UK, 8–10 February, pp. 3–23. Springer, London, UK.