


# Mixed Feelings About Mixed Criticality

Reinhard Wilhelm

Informatik, Universitaet des Saarlandes, Saarland Informatics Campus

Saarbruecken, Germany

wilhelm@cs.uni-saarland.de

 <https://orcid.org/0000-0002-1825-0097>

---

## Abstract

I point to some challenges for WCET analysis offered in the transition to integrated mixed-criticality systems (MCSs) and to multi-core platforms, claim that proposed certification standards are inadequate, show that the MCS model heavily used by the scheduling community is fraught, and clarify why the traditional abstract interface between WCET analysis and schedulability analysis is obsolete.

A central point is the insistence on sound approaches. I give a detailed account of how the most rigid certification procedures, those of the avionics domain, are satisfied, to defend the validity of my claims.

**2012 ACM Subject Classification** Software and its engineering → Real-time systems software

**Keywords and phrases** WCET analysis, mixed criticality systems, multi-core platforms, scheduling, schedulability

**Digital Object Identifier** 10.4230/OASIScs.WCET.2018.1

**Category** Invited Paper

**Funding** Funding for our research was obtained from Deutsche Forschungsgemeinschaft under Collaborative Research Centers 124 and AVACS and the Project Daedalus in the European 5th Framework Programme.

**Acknowledgements** I would like to thank Rob Davis, Markus Pister, Gernot Gebhard, and Jan Reineke for support in writing this paper. Rob Davis tried to convince me of the value of the Vestal model of MCS. Markus Pister enlightened me about certification processes relating to safety-critical systems. Gernot Gebhard let me use an illuminating figure from his PhD thesis. Jan Reineke provided valuable comments on earlier versions of this article.

## 1 Introduction

The general setting for WCET analysis is that a set of hard real-time tasks is to be executed on a given hardware platform. Being hard real-time tasks means having associated deadlines within which they have to finish their execution. *Timing Verification* has to verify that these timing constraints are satisfied. Traditionally Timing Verification is split into a *WCET analysis*, which determines upper bounds on the execution times, and a *schedulability analysis*, which takes these upper bounds and attempts to verify that the given set of tasks when executed on the given platform will all respect their deadlines.

There are two strong trends in the embedded-systems industry, the transition from single-core to multi-core execution platforms and the transition from federated systems to integrated systems comprising components with different levels of criticality.



© Reinhard Wilhelm;

licensed under Creative Commons License CC-BY

18th International Workshop on Worst-Case Execution Time Analysis (WCET 2018).

Editor: Florian Brandner; Article No. 1; pp. 1:1–1:9

OpenAccess Series in Informatics



OASIS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The criticality level (aka Safety integrity level (SIL)) of a component is derived from the impact of a failure of the component on the functioning of the system. It determines the size of the effort to deliver assurance of the correct functioning of the component. A mixed criticality system (MCS) is one that has two or more distinct criticality levels. Up to five levels exist in the standards, e.g. the IEC 61508, DO-178B and DO-178C, DO-254 and ISO 26262.

For architectures with instructions that had constant execution times, WCET analysis methods using *Timing Schemata* [14] were the method of choice. Timing schemata describe how (bounds on) the execution times of a programming-language construct were composed from the (bounds on) the execution times of its components. These methods would thus do structural induction over the structure of a program and determine bounds for ever bigger parts of the program.

Performance-enhancing architectural components such as caches, pipelines, and speculation made previous methods for WCET analysis using *Timing Schemata* [14] obsolete. Execution times do not compose any longer because instruction execution-times are now dependent on the execution state in which they were executed. In the composition  $A;B$  the execution time of statement  $B$  depends on the execution state produced by statement  $A$ . The variability of execution times grew with several architectural parameters, e.g. the cache-miss penalty and the costs for pipeline stalls and for control-flow mis-predictions.

## 1.1 The Central Idea – Proving Safety Properties

We started off to solve the WCET problem for architectures with state-dependent execution times. Let me describe the central idea behind the *microarchitectural-analysis* phase in our WCET-analysis method [18], first in a conceptual way, i.e. not quite like it is implemented, later closer to how it is implemented:

- Define any architectural effect that causes an instruction to execute longer than its fastest execution time as a *Timing Accident*. Typical such timing accidents are cache misses, pipeline stalls, bus-access conflicts, or branch mis-predictions. Each timing accident is associated with a *Timing Penalty*. Timing penalties may be constant, but may also be execution-state dependent.

The property that an instruction will not cause a particular timing accident is then a safety property. The occurrence of a timing accident thus violates a corresponding safety property.

- Use an appropriate method for the verification of safety properties to prove that for the instructions in the program some of the potential timing accidents will never happen. The goal is to prove as many of such safety properties as possible. Conceptually, the safety properties shown to hold could be used to reduce the worst-case execution-time bound for an instruction, which a naive, sound WCET analysis would have to assume, by the cost for the excluded timing accidents. In practice, pipeline analysis drives a cycle-wise transition, which considers the abstract execution state, e.g. makes no transition under a cache miss if a cache miss can be excluded.
- Prove these safety properties by abstract interpretation (AI) [7] in the following way: Use AI to compute invariants at each program point, in our case an upper approximation of the set of execution states that are possible when execution reaches this program point. Derive the above mentioned safety properties, that certain timing accidents will not happen, from these invariants. For example, AI computes abstract cache states at each program point, which represent the sets of concrete cache states that may reach this program point. The abstract cache states are used to classify memory accesses at each

program point as definite hits or misses. Predicted cache hits are then used to prove that the timing accident, this memory access will miss the cache, will never happen [10].

This method for the micro-architectural analysis was the main innovation that made our WCET analysis work for real-life architectures and scale to industrial-size software [9].

Now follows the description of the microarchitectural analysis that is closer to the implementation. Driver of this analysis is the pipeline analysis [15]. It goes through the instruction stream, instruction by instruction, and executes the current instruction on the current abstract execution state. This abstract execution state contains uncertainty, i.e. misses some components. Transitions to all potential successor states are performed whenever the transition to the next state depends on such a missing part of the state. The timing contributions of these transitions are accumulated until an instruction can be retired. In the end upper bounds on the execution times of basic blocks are obtained that are coefficients in an Integer Linear Program representing the control flow of the program [18].

We currently experience two significant developments in the safety-critical embedded-systems industry that are of concern to the WCET-analysis domain, the introduction of multi-core execution platforms and the integration of applications of different criticalities on such platforms. As we will later see, the clean interface between schedulability and timing analyses becomes obsolete as soon as multi-core architectures with shared resources are used for the implementation of hard real-time systems, and the (extremely productive) scheduling community has adopted a system model, ignoring fundamentals of WCET analysis.

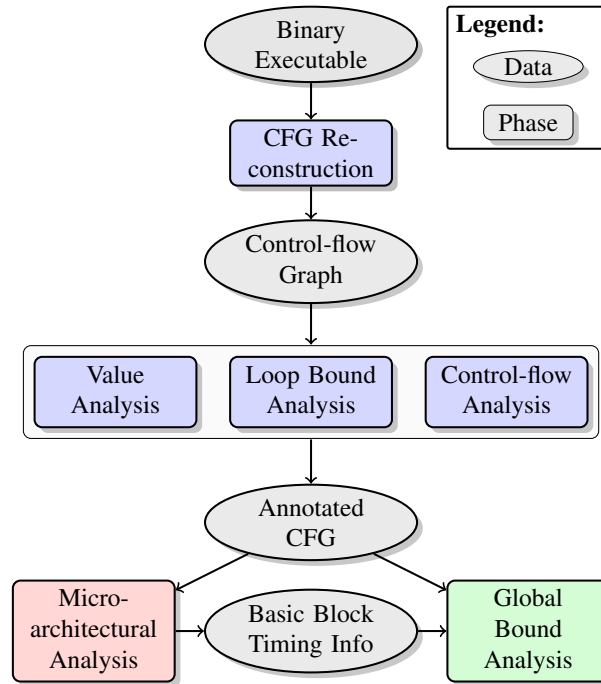
## 1.2 Terminology

We consider only sound WCET-analysis methods. *Soundness* means that a method and associated tool will always produce *conservative* WCET estimates, i.e. estimates that will never be exceeded in any execution. Being conservative is a Boolean property. Unfortunately, *conservative* is often used as a metric property, *more conservative* meaning *less precise*. However, calling results of an unsound method conservative is a misnomer. The really meant, other dimension, in addition to soundness, is *accuracy*. Accuracy of some WCET estimate, obtained by a sound method, expresses the degree of over-estimation, the difference between a WCET estimate and the real WCET. It does not make sense to talk about the accuracy of an unsafe estimate or an unsound method. In case of an unsound method it is not even clear whether a "more conservative" estimate moves towards the real WCET from below or is larger than the real WCET and moves further away from it.

WCET analysis can be seen as the search for a longest path in the state space spanned by the program under analysis and by the architectural platform. Most real-time software is written as to guarantee termination. Its state space can thus be easily abstracted to a finite abstract state space, which is still too large to be exhaustively explored. We can, therefore, not expect to identify the real WCET, but only safe upper bounds to all execution times, which we will call WCET estimates. (Safe) over-approximation is used in several places. In particular, an abstraction of the execution platform is employed by the WCET analysis. How to convince oneself (or the certification authorities) of the correctness of this architectural model is the subject of the next section.

## 2 Certification

The claim that our WCET-analysis tools produce safe results is a strong one and often disputed by some proponents of unsound WCET-analysis methods. Their argument is, to develop an error-free instantiation of the, in principle, sound WCET-analysis technology



■ **Figure 1** The architecture of the aiT tool.

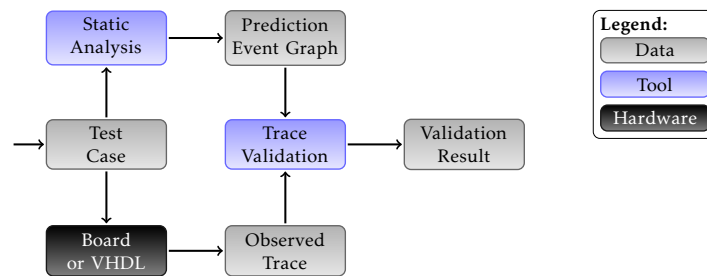
is so difficult, that one might use a simpler unsound method in the first place. The main complaint is the complexity of the abstract architectural models. So, what is the basis for our claims?

### Tool Qualification according to DO178

Let us start with a description of the safety standards and the tool-qualification processes of the avionics industry, which are the most rigid of the safety-critical industries. Certification of avionics systems is regulated by the international standard DO-178C [1]. WCET-analysis tools fare under *verification tools*. Verification tools have no overly rigid certification requirements, unlike *development tools*. They require a specification of the tool functionality, from which several levels of requirements are derived. DO-178C exhales a test-based spirit. Most of the qualification is test based, requiring some coverage criteria to be observed. However, note that in case of a static verification tool, test coverage means something different from the usual interpretation as, e.g. coverage of the program control flow. At analysis time, a static-analysis tool analyses all paths and does not need coverage criteria for its analysis. It is the ISA and the set of paths through the execution platform that need to be covered. Huge sets of test traces in qualification suites are used at tool-qualification time to cover the sets of paths through the execution platform.

Certification becomes more challenging through DO-333, the Formal-Methods Supplement to DO-178C. It asks for a statement that a formal method including the underlying theory is *adequate* for solving the corresponding verification problem. This introduces and enforces *soundness* of the methods and tools.

Several component analyses in the tools are instances of abstract interpretation [7], a scientific method with a strong underlying theory, relating analysis results to semantic properties of analyzed programs. Value analysis and control-flow analysis, c.f. Figure 1



■ **Figure 2** Trace Validation according to [11]. The instruction sequences together with the generated prediction graphs, annotated by state and timing information are part of the Qualification Support Kit.

are more or less standard abstract interpretations, the difference is that these analyses are performed on the binary level and not on the source level. Still, adequacy of these analyses is easily accepted. The instantiation of the abstract-interpretation framework for the microarchitectural analysis, however, is far from trivial. In particular, each contains an abstraction of the execution platform. How does one make sure that such an abstraction is conservative? The European Aviation Safety Agency (EASA) is strict on this issue. It has accepted AbsInt’s aiT as a validated WCET-analysis tool for several time-critical subsystems in the Airbus A380 and A350 planes.

## Trace Validation

EASA requires the tool user to perform a tool qualification. As written above, the most complex part of the WCET-analysis tool is the abstract architectural model. Therefore, the most complex task in tool qualification is the validation of this abstract architectural model. It is done by *Trace Validation*. The tool user may ask the tool provider to support them by providing a *Qualification Support Kit (QSK)* containing the abstract architectural model and sets of test traces, annotated with timing information. The abstract model is used as a generator of event traces. Typically, only events that can be externally observed are generated and thus contained in traces in the prediction graph. Several (hand-written) instruction sequences, *test cases* according to Figure 2, are run through this abstract model, each producing a graph of traces, the so-called *Prediction Graph* [11].

In trace validation, an instruction sequence is executed on the actual hardware. Interrupts are used to stop execution at each desired execution cycle. This way, the execution of instruction sequences are extended cycle by cycle to observe actual execution states and execution times. Whatever machine information can be read out is used. The observed trace, the reached execution state and the consumed time are checked for containment in the prediction graph. The predicted execution time may be larger than the observed execution time, but never smaller. Some interesting components of the architectural state, e.g. the cache state, are not directly observable. These need to be indirectly observed through executions that are forced to lead to cache hits and cache misses. A tremendous effort is invested to cover both all instructions and all architectural components, essentially by triggering many different initial architectural states.

In the case of the AbsInt static WCET tool, aiT, the validation suite may contain several thousand event traces, even for a simple DLX-like architecture like the ARM Cortex-M4.

**Testing in the Operating Environment**

In addition, DO-178 asks the user of a tool to be qualified to test the tool in their operating environment. This includes testing it on representative user code, besides testing it possibly on synthetic examples. In model-based design processes, which are quite common in the safety-critical embedded-systems domain, this is often done by exhaustively testing patterns used by the code generators.

**3 Multi-Core Architectures**

WCET analysis for single-core architectures is theoretically understood and practically solved. The significance of timing-predictability of execution platforms is recognized, but has left few traces in the architectural domain, a notable exception being the Kalray MPPA [8]. The transition of the embedded systems industry to multi-core platforms presented new challenges by increasing the complexity of WCET analysis considerably. In general, all possible interleavings of the concurrently executed tasks have to be analyzed, since different interleavings may lead to different execution times. The reason was is the interaction on shared resources of tasks executing on different cores [2].

The interference on shared resources of tasks running on different cores invalidates the traditional interface between WCET analysis and schedulability analysis, which is the following: WCET analysis determines an upper bound on the execution times of a task, and schedulability analysis uses this bound as input. However, different schedules on the different cores lead to different interactions on the shared resources, and in consequence, to different execution times of the tasks. So, the WCET estimate determine the schedules, and the schedules influence the execution times. This fact is ignored by quite a few people working on multi-core scheduling.

A position paper on the use of multi-core platforms in future avionics systems [6], written by an international consortium, recognizes that the interference on shared resources makes the traditional spatial and temporal partitioning methods required by ARINC 653 problematic. They require *robust partitioning* of the co-executing tasks to allow separate WCET determination. The relevant necessary condition for robust partitioning reads, *Software partitions cannot consume more than their allocations of shared resources*. This formulation, while applicable to bandwidth resources like buses, ignores the important differences between *storage resources* and *bandwidth resources*. Caches are typical storage resources. Analyzing shared caches is particularly challenging. It is clear that the cache state, and therefore also the cache-miss rate and the execution time, depend on the particular interleaving of the executions of different co-executed tasks. Buses are typical bandwidth resources. Competition for this resource is resolved by bus protocols, which then influence, for example, the memory-access time. Bus protocols become part of the WCET analysis.

In case the requirement for robust partitioning is violated the position paper asks for *mitigation* by the developer. The only problem is that it remains unclear how such mitigation could look like. [19] gives a survey of promising approaches for achieving robust partitioning.

**4 Mixed Feelings about Mixed Criticality**

Steve Vestal [17] has proposed a model of mixed-criticality systems for schedulability analysis. This model is based on a conjecture that *the higher the degree of assurance required that actual task execution times will never exceed the WCET parameters used for analysis, the larger and more conservative the latter values become in practice*. The survey [5] of mixed-criticality

systems by Burns and Davis adopts the same assumption, *A key aspect of MCS is that system parameters, such as tasks' worst-case execution times (WCETs), become dependent on the criticality level of the tasks.* These authors, however, seem to be skeptical about this assumption: *Although it is reasonable to assume confidence increases (i.e. uncertainty decreases) with larger estimates of worst-case execution time, this may not be universally true. It would certainly be hard to estimate what increase in confidence would result from, say, a 10% increase in all Cs.* It is illuminating that both articles do not mention soundness, as if it were of no concern in the safety-critical systems domain.

I see no inherent reason why higher criticality should entail higher WCET estimates. It seems that this assumption is intuitively based on the assumption that WCET estimates are determined by measurement; higher criticality levels require higher assurance, and higher assurance is achieved by performing increasing sets of tests. Since WCETs monotonically increase with increasing the set of tests – an observed WCET does not disappear, when more tests are added – more tests can, in fact, not produce lower WCET estimates.

#### 4.1 Exploitation of Hardware Resources

Another motivation is an assumed higher exploitation of the hardware performance: Let us assume that by extensive measurement the Maximum Observed Execution Time (MOET) of the high-criticality task  $T_h$  is found to be substantially less than the WCET estimate,  $C_h(HI)$  provided by a sound tool. This MOET may be considered as an intermediate (low-criticality) budget,  $C_h(LO)$ . Some low criticality software,  $T_l$ , with no strong guarantees, which will be run on the same hardware platform, might have a MOET of  $C_l(LO)$ . The scheduler may drop or degrade the low-criticality task in the event that either  $T_l$  exceeds its MOET of  $C_l(LO)$  or  $T_h$  exceeds its MOET of  $C_h(LO)$ . Since the high-criticality task must execute and must meet its timing constraints, the overall performance required of the system is given by  $\max(C_l(LO) + C_h(LO), C_h(HI))$ , which may be substantially less than  $C_l(LO) + C_h(HI)$ . The strength of this motivation, of course, depends on the size of  $C_h(HI) - C_h(LO)$ , i.e. on the amount of over-estimation. There are a few publications documenting the amount of over-estimation, see [16]. Between 15 and 25 % over-estimation were observed on real Airbus code. Of course, the amount of over-estimation depends on many factors, in particular on the timing predictability of the execution platform [12, 13, 4].

#### 4.2 Schedulability Analysis

Vestal thus starts with the assumption that different WCETs are associated with different criticality levels of a task, the higher the criticality level, the higher the WCET estimate, and then proposes to use two different versions of preemptive fixed-priority (PFP) scheduling with deadline-monotonic priority assignment; tasks with smaller deadlines get higher priority.

The first approach attempts to solve the problem that low-criticality tasks with shorter deadlines than higher-criticality tasks would receive higher priorities. By cutting the longer execution times of higher-criticality tasks into short time slices they receive higher priorities. The scheduling algorithm then is able to use time left over by higher-criticality tasks not exhausting their WCET estimate for lower-criticality tasks. So far so good! No treatment is dedicated to the case that the high-criticality task exceeds its WCET estimate and in consequence its deadline. This is possible since measurement-based analyses are not guaranteed to produce safe upper bounds.

Vestal's second approach uses Audsley's priority-assignment algorithm [3] in a setting with WCET bounds increasing with criticality level, for which it was not originally described. I assume that the algorithm can be adapted to work for a setting with different WCET estimates associated with different criticalities.



### 4.3 Consequences of Ignoring Sound Approaches

So, Vestal's schedulability test is only sound with respect to correct WCET estimates and will, if given incorrect estimates, accept task sets whose high-criticality tasks may at run time exceed their deadlines. The scheduling community on the one hand would claim that their algorithms are sound, but gladly accepts input produced by unsound methods, which invalidates the overall correctness claim. However, his model has been and still is the underlying model for most scheduling research on mixed-criticality systems. [5] lists almost 200 publications.

---

#### References

- 1 RTCA/DO-178C Software Considerations in Airborne Systems and Equipment Certification, 2013.
- 2 Andreas Abel, Florian Benz, Johannes Doerfert, Barbara Dörr, Sebastian Hahn, Florian Hauptenthal, Michael Jacobs, Amir H. Moin, Jan Reineke, Bernhard Schommer, and Reinhard Wilhelm. Impact of Resource Sharing on Performance and Performance Prediction: A Survey. In Pedro R. D'Argenio and Hernán C. Melgratti, editors, *CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, volume 8052 of *Lecture Notes in Computer Science*, pages 25–43. Springer, 2013. doi:10.1007/978-3-642-40184-8\_3.
- 3 Neil Audsley. Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times. Technical Report 164, Department of Computer Science, University of York, November 1991.
- 4 Philip Axer, Rolf Ernst, Heiko Falk, Alain Girault, Daniel Grund, Nan Guan, Bengt Jonsson, Peter Marwedel, Jan Reineke, Christine Rochange, Maurice Sebastian, Reinhard von Hanxleden, Reinhard Wilhelm, and Wang Yi. Building timing predictable embedded systems. *ACM Trans. Embedded Comput. Syst.*, 13(4):82:1–82:37, 2014. doi:10.1145/2560033.
- 5 A. Burns and R. Davis. Mixed criticality systems-a review. Technical report, Department of Computer Science, University of York, 2013.
- 6 Certification Authorities Software Team. *Multi/core processors*, CAST-32A edition, November 2016. position paper.
- 7 Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In Robert M. Graham, Michael A. Harrison, and Ravi Sethi, editors, *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 238–252. ACM, 1977. doi:10.1145/512950.512973.
- 8 Benoît Dupont de Dinechin, Duco van Amstel, Marc Poulhiès, and Guillaume Lager. Time-critical computing on a single-chip massively parallel processor. In Gerhard Fettweis and Wolfgang Nebel, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, pages 1–6. European Design and Automation Association, 2014. doi:10.7873/DATE.2014.110.
- 9 C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and Precise WCET Determination for a Real-Life Processor. In *EMSOFT*, volume 2211 of *LNCS*, pages 469–485, 2001.
- 10 Christian Ferdinand and Reinhard Wilhelm. Efficient and Precise Cache Behavior Prediction for Real-Time Systems. *Real-Time Systems*, 17(2-3):131–181, 1999.
- 11 Gernot Gebhard. *Static timing analysis tool validation in the presence of timing anomalies*. PhD thesis, Saarland University, 2013. URL: <http://scidok.sulb.uni-saarland.de/volltexte/2013/5558/>.



- 12 Reinhold Heckmann, Marc Langenbach, Stephan Thesing, and Reinhard Wilhelm. The influence of processor architecture on the design and the results of WCET tools. *IEEE Proceedings on Real-Time Systems*, 91(7):1038–1054, 2003.
- 13 Jan Reineke, Daniel Grund, Christoph Berg, and Reinhard Wilhelm. Timing predictability of cache replacement policies. *Real-Time Systems*, 37(2):99–122, 2007. doi:10.1007/s11241-007-9032-3.
- 14 Alan C. Shaw. Deterministic timing schema for parallel programs. In V. K. Prasanna Kumar, editor, *The Fifth International Parallel Processing Symposium, Proceedings, Anaheim, California, USA, April 30 - May 2, 1991.*, pages 56–63. IEEE Computer Society, 1991. doi:10.1109/IPPS.1991.153757.
- 15 Stephan Thesing. *Safe and Precise WCET Determinations by Abstract Interpretation of Pipeline Models*. PhD thesis, Saarland University, 2004.
- 16 Stephan Thesing, Jean Souyris, Reinhold Heckmann, Famantanantsoa Randimbivololona, Marc Langenbach, Reinhard Wilhelm, and Christian Ferdinand. An Abstract Interpretation-Based Timing Validation of Hard Real-Time Avionics Software. In *2003 International Conference on Dependable Systems and Networks (DSN 2003), 22-25 June 2003, San Francisco, CA, USA, Proceedings*, pages 625–632. IEEE Computer Society, 2003. doi:10.1109/DSN.2003.1209972.
- 17 Steve Vestal. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*, pages 239–243. IEEE Computer Society, 2007. doi:10.1109/RTSS.2007.47.
- 18 Reinhard Wilhelm, Sebastian Altmeyer, Claire Burguière, Daniel Grund, Jörg Herter, Jan Reineke, Björn Wachter, and Stephan Wilhelm. Static Timing Analysis for Hard Real-Time Systems. In Gilles Barthe and Manuel V. Hermenegildo, editors, *Verification, Model Checking, and Abstract Interpretation, 11th International Conference, VMCAI 2010, Madrid, Spain, January 17-19, 2010. Proceedings*, volume 5944 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2010. doi:10.1007/978-3-642-11319-2\_3.
- 19 Reinhard Wilhelm, Jan Reineke, and Sven Wegener. Keeping Up with Real Time. In Umut Durak, Juergen Becker, Sven Hartmann, and Nikolaos S. Voros, editors, *Advances in Aeronautical Informatics: Technologies Towards Flight 4.0*. Springer, 2018.