Usability Aspects of WCET Analysis*†

Jan Gustafsson
School of Innovation, Design and Engineering, Computer Science Department,
Mälardalen University, Västerås, Sweden
jan.gustafsson@mdh.se

Abstract

Knowing the program timing characteristics is fundamental to the successful design and execution of real-time systems. A critical timing measure is the worst-case execution time (WCET) of a program. Often, timing analysis in industry is done by measurements. Recently, tools for deriving WCET estimates have reached the market.

With more widespread use of WCET tools in industry, the usability *aspects of these tools will be of growing importance. In this paper we discuss usability using the results of the WCET Challenge 2006, which was the first event that compared different WCET tools using the same set of benchmarks. Another source of input to the discussion are experiences from industrial case-studies of WCET tools.*

Finally, we point out some areas for future research and development for WCET analysis methods and tools.

1 Introduction

A *program timing analysis* obtains information about the execution time characteristics of a program. The *worst-case execution time* (WCET) of a program is a key timing measure. The WCET measure is based on the assumption that the software is executed in isolation on an undisturbed processor. Effects outside of the code that may affect this timing, e.g., interrupts from the hardware, the operating system or other tasks, can be accounted for in subsequent analysis steps, e.g., in scheduling analysis.

Figure 1 shows how measurements relate to the set of possible program executions and timings. The example program has a variable execution time, and the darker curve shows the probability distribution of its execution time. Its minimum and maximum are the BCET (best case execution time) and WCET respectively. The lower gray curve shows the set of *actually* observed and measured execution times. Its minimum and maximum are the *minimal measured time* and *maximal measured time* respectively.

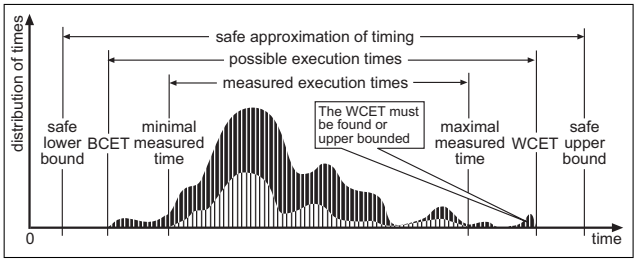

Figure 1. Execution time estimates

Reliable safe upper bounds of the WCET are important when designing and verifying embedded systems and real-time systems, especially for systems used to control safety critical products and processes. For less safety-critical systems, exact WCET estimates are not always required. In some applications, only limited parts of the system software are time critical and in need of timing analysis.

Timing and WCET analyses are today performed using three main methodologies:

- *Measurements* are done executing the program on the actual target hardware.
- *Static WCET analysis* avoids executing the program. It derives a WCET estimate by static analysis of the program and the timing properties of the used hardware.
- *Hybrid analysis* methods (or *measurement-based methods*) combine measurements and static analysis.

The state-of-the-practice in industry is still to use measurements, but recently commercial static and hybrid WCET analysis tools have been introduced and are used to an increasing extent in industry. The goal of these WCET tools is to provide a WCET estimate for a certain piece of code that is *safe*, i.e., equal to or larger than the real WCET[1]. To be useful, especially in a system with limited resources, the estimate should also be *tight*, i.e., as close to the real WCET as possible.

* Supported by KK-foundation (www.kks.se), grant 2005/0271.
† ARTIST2 European Network of Excellence (http://www.artist-embedded.org/artist/-Cluster-Compilers-and-Timing-.html) provided travel support.

[1] It should be noted, while the static analysis methods aims at giving an *absolute* WCET estimate, the hybrid method (RapiTime) gives a *probabilistic* WCET estimate. See also Section 2.3.

As the rest of this paper will point out, there are a number of other aspects that concern the practical use of WCET tools. There is often a lot of work to do before a useful result is found. Therefore, the usability aspect of these tools is an important issue.

The main contributions of this paper are:

- We present a number of important usability aspects of WCET methods and tools.
- We discuss how these aspects are handled in some of the current methods and tools.

The rest of this paper is organized as follows: Section 2 gives a short overview of timing analysis methods. Section 3 describes the sources of information for this paper. Section 4 presents usability aspects of WCET tools, and Section 5 gives some conclusions and ideas for future work.

2 Overview of WCET Analysis

For a more detailed overview of the different methodologies and references to relevant papers, see, e.g., [6, 28].

2.1 Measurements.

The program is executed on the target hardware with worst-case input (if it is known), and the execution time is measured. If the worst-case input is not known, the program is executed many times with different inputs, the execution time is measured for each test run, and finally, the longest time is selected. Measurements on the actual hardware avoids the need to construct a hardware model.

Measurement-based methods can be divided into *software-based* and *hardware-based* methods.

Software-based methods can use timing functions provided by operating systems or programs provided by tool vendors designed specifically for execution time measurement. These methods often introduce a probe effect, i.e., the measurements themselves add to the execution time of the analysed program. This problem can be alleviated by simply letting the added measurement code (and thus the extra execution time) remain in the final program.

An alternative is to use hardware measurement tools with no or a very small probe effect, e.g., oscilloscopes, logic analyzers, and in-circuit emulators. The Nexus interface is sometimes used for timing measurements.

It should be noted that each measurement runs through only *one* path of the program. For programs with a single path, or a few paths, measurements might be feasible. However, for most programs, the possible execution paths are too many to do exhaustive testing, and the worst-case input is not known. This means that the measured times often underestimate the WCET.

In general, measurements are suitable for less time-critical software, where an approximate WCET estimate may be sufficient. If measurements are used for time-critical software, where a safe WCET must be known, extreme caution must be taken to avoid the problem of underestimation mentioned above.

An alternative to do measurements on the actual hardware is to use cycle-accurate simulators. However, these are scarce, and they also introduce the question of correctness of the results.

2.2 Static WCET analysis.

An alternative technique to determine WCET estimates is by *static WCET analysis*. Instead of running the program, it derives a WCET estimate by statically analysing the timing properties of the program. Given that inputs and analyses are correct, such a tool will derive a *safe* estimate, i.e., that is larger than or equal to the WCET.

Static WCET analysis is usually divided into three phases: a *flow analysis* where information about the possible program execution paths is derived, a *low-level analysis* where the execution time for atomic parts of the code (e.g., instructions, basic blocks or larger code sections) is decided from a model of the target architecture, and a final *calculation* phase where the derived flow and timing information are combined into a resulting WCET estimate.

Loop bounds must be known in order to derive finite WCET estimates. These bounds can either be given as manual annotations, or be calculated automatically. Much of flow analysis research has studied automatic loop bound calculation. Some flow analysis research has aimed to identify *infeasible paths*, i.e., paths which are executable according to the control-flow graph structure, but not feasible when considering the semantics of the program and possible input data values. Such information can yield a tighter WCET estimate.

The main issue for low-level analysis is the complex behaviour of modern hardware, with features like pipelines, caches, branch prediction, and out-of-order execution. The timing behavior of multi-core processors will be the next big issue for research. Models or simulators of the hardware are used in the low-level analysis. This eliminates the need of having the actual hardware available, but a (safe) timing model of the hardware must be developed, something which can be very complicated. Some research projects study how to simplify the process of hardware timing model development.

Due to the complexity of today's software and hardware, both flow- and low-level analysis may yield over-approximations, e.g., reporting too many paths as feasible or too large timings to instructions. Thus, the calculation may give a pessimistic WCET value.

Today, two static WCET tools are commercially available: aiT [1] and Bound-T [26]. Several research prototypes have been developed, including Chronos [4], the Florida State University tool [11], Heptane [12], OTAWA [17], SWEET [10], and TimeBounder [3].

2.3 Hybrid WCET analysis.

Hybrid analysis methods combine measurements and static analysis. The tools use measurements to extract timing for smaller program parts, and static analysis to deduce the final WCET estimate from the program part timings. Examples of hybrid tools are RapiTime [20], SymTA/P [23] and MTime [27].

There is a possibility that the hybrid methods underestimate the WCET, since the WCET estimate is based on measurements, and measurements may exclude the worst case path. The selection of test cases to reach the best path coverage is therefore crucial when using hybrid methods. An advantage of the hybrid approach may be that selection of test cases and control of coverage are well-known techniques in software engineering.

Actually, hybrid methods may also overestimate the WCET, since measurements from mutually exclusive parts of the program may be combined in the final WCET.

RapiTime is able to either analyse source code, adding instrumentation points on the source code level, or, otherwise use binary readers and instrument the generated code. In RapiTime, measurements are combined into *execution profiles*, from which the probability of an execution to exceed a certain time budget can be found.

The hybrid approach seems to be suitable for systems where an absolutely safe WCET is not strictly necessary, or for complex processors where a processor model is not available and may be vary hard to develop.

3 Sources of Information

The main inputs to this paper will be presented in more detail in this section.

WCET Challenge 2006. The WCET Challenge 2006 [8, 24] was the first event that compared different WCET tools using the same set of benchmarks. The WCET Tool Challenge used programs from the Mälardalen WCET benchmark suite [21] and the PapaBench benchmark [18].

The purpose of the WCET Tool Challenge was to be able to study, compare and discuss the properties of different WCET tools and approaches, to define common metrics, and to enhance the existing benchmarks. The WCET Tool Challenge was designed to find a good balance between openness for a wide range of analysis approaches, and specific participation guidelines to provide a level playing field. The WCET Tool Challenge concentrated on the following three aspects of WCET analysis: *flow analysis*, *user interaction*, and *performance*.

Five WCET tools entered the Challenge. Of these, two were commercial (aiT [1] and Bound-T [26]) and three were research prototypes (Chronos [4], MTime[2] [27] and SWEET [10]). There is a short description of each of these tools in [8].

The main conclusions from the Challenge was:

- The tests were a real challenge to the participating WCET tools. We had a success range from 76% to 100% in terms of how many of the benchmark programs that were analyzable by a certain tool[3].
- The main result of the Challenge was not a measure of the tightness of the WCET results. The results rather focused on the usability aspects - the trouble connected with providing inputs and running the tools.
- The tests clearly pointed out problems existing in the tools, in the benchmarks and the used compilers.
- Several bugs in both the tools and the benchmarks were corrected during the Challenge.
- Full automation was not reached; the best result were 88% of the benchmarks (the automation rate calculated as the ratio of the number of automatically analyzed programs to the number of all analyzable ones).
- Most of the tools found more than half of the loop bounds automatically. Two tools found infeasible paths automatically.
- Actual WCET estimates could not be compared since the tools supported different processors and compilers.
- The quality of WCET estimates was hard to judge for all tools but aiT, since aiT was the only tool to provide worst case measurements for some of the benchmarks. Chronos provided simulated values that indicated the possible size of overestimation.

WCET Tool Case Studies. Experience reports from the use of WCET tools in industry are rather scarce. There are some reports on the use of commercial WCET tools for analyzing codes for space applications [13, 14, 22], in avionics software [7, 25], and in automotive software [16].

The WCET group at Mälardalen University have performed a number of case studies [9]. These studies include WCET analysis of the OSE operating system, code controlling welding equipment, communication software in cars, and transmission code.

The case studies referenced above mainly describe work with static WCET tools. One interesting exception is the pair of Master's theses [5, 29]. In these, the same code was analysed using both measurements (with different methods) and static analysis tool (aiT), and results were compared.

To the author's knowledge, there is yet no experience report written for hybrid tools. Also, no hybrid tool entered the WCET Challenge 2006. Therefore, this paper will not cover experiences of such tools.

The results and experiences from the sources above are used in the discussion in the rest of the paper. For simplicity, there will be no references to individual sources in the text.

[2] Since the MTime tool did not support function calls, and all benchmarks contain such calls, no results from MTime were available this time.

[3] MTime excluded.

4 Usability Aspects of WCET Tools

This section gives an overview of usability aspects of WCET tools. First, some basic considerations are discussed, followed by a more detailed discussion of usability aspects for different types of tools.

4.1 Basic Considerations

Facing the task of estimating the WCET of some piece of code, there is a set of immediate questions that must be answered. They have to do with type of processor, selection of code to analyse, requirements of the result, selection of the most suitable method, etc. Once these questions are answered, the next step will be to proceed with the analysis using the chosen methods and tools.

Which processor is used? Not all processors are supported by the existing tools. An important issue is that a model of the processor has to be built for static analysis tools, which means that such models may not be available for uncommon, new, or very complex processors. For measurements and hybrid methods, no such model is required, since the timing values are measured on the hardware itself.

How complex is the hardware? Many of today's processors are complex and non-deterministic with pipelines, caches, out-of order execution, and other advanced features that speeds up the average execution, but make the timing behaviour more dynamic and especially the worst case execution time harder to calculate. The more complex the processor is, the more complex is static WCET calculation, and the larger the risk for large overestimations. For measurements and hybrid methods, the complexity of WCET calculations is mitigated, but the large variations of execution time may still lead to large overestimations.

Is the hardware available? Sometimes, hardware and software are developed in parallel, making it impossible to measure timing on the target hardware. In such cases, static analysis is advantageous, assuming a model of the processor to be used is available. However, it is sometimes the case that accurate simulators of the processor are available before fully functional silicon is. Measurements and hybrid methods can then use these simulators.

What timing value is really wanted?

- *Is a safe WCET estimate required?* If so, then static analysis probably is the right way to go.
- *Is an approximate WCET estimate sufficient?* That may be the case, if the system is a soft real-time system, or if the system can tolerate occasional overruns. In that case measurement-based timing values can be sufficient. Hybrid tools are another option, sometimes giving more detailed information about the result than just a single value (see Section 2.3).
- *Is more information than just a single WCET wanted?* Today's static and hybrid tools provide useful extra information, for example the timing for individual functions, identification of bottlenecks, and information about the longest path. Sometimes, parametrical values are of interest, i.e., the WCET as a function of, e.g., the inputs to the analysed code. Be aware of that most of today's timing analysis methods most often support the calculation of one value only. On-going research studies how to find parametrical WCET values.

What is the size of the code? Some static tools may have long analysis times for large programs. These tools often support precision level adjustment, and sometimes faster analysis can be achieved to the price of lower precision, which may be acceptable at the beginning of a project.

What part of the code should be analysed? Is it the whole task, or part of it? Do you differ between running modes, e.g., the start-up phase of the system and the system during normal operation? Sometimes you want to exclude some paths since they represent program states which are not of interest for WCET, but take much longer to execute than normal operation. Error handling is one such example. Prepare for more work the more detailed the restrictions are.

What is the structure of the system? Does the system use a real-time operating system (RTOS) or not? Is the code task-oriented? It might affect the analyzability of a system when using static or hybrid methods. Systems with many small and well-defined tasks, scheduled by a strict priority RTOS or a time-triggered schedule, are typically easier to analyze than monolithic programs based on an infinite main loop.

How is the code written? This also matters when using static or hybrid methods. Simple and straight-forward code, written with WCET analysis in mind, may impose less requirements on the WCET analyzer, which makes them faster to analyze, and yields safer results, than complex code. Examples of complex troublesome constructs (or even impossible for some WCET tools) to analyze is the use of loops with complex exit conditions and deep nesting, recursion, dynamic memory, function pointers, and unstructured code.

Much code today is generated from model-based tools like TargetLink, which means that the code available is often not really readable for the programmer. Adaptation of WCET tools sometimes have to be made for these tools. On the other hand, the code can be generated so that WCET analysis is simplified, e.g., complex constructs can be avoided.

There is a coding style, single path programming [19], where algorithms are selected so that the resulting program contains just one path. In this case, flow analysis is trivial, and the WCET can be calculated by doing a low-level

analysis for just one path (or simply run the program and measure the time).

Is all code available? To do meaningful measurements, a complete and executing program has to be available. When using static analysis, not finished parts may be replaced by stubs to get allow compilation and linking. In aiT, parts of the program can be excluded from analysis and be assigned a WCET by manual annotations.

Is the source code available? Measurements do not require source code. Some static and hybrid tools (e.g., aiT, Bound-T, and RapiTime) are able to analyse executable code without having access to the source code. They use binary readers that decode the executable code and reconstruct the control flow graph of the program. However, for many actions, things become easier if you have access to the source code. For example, when using static methods, adding manual annotations is much more convenient on source code level.

WCET tools that require source code (e.g., SWEET) are of course dependent of source code availability. Since low-level analysis must be made at executable code level, these tools are, in this case also dependent on the compiler. Libraries linked to the code may not be available as source code, which imposes a problem to such tools.

What programming language is used? WCET tools that analyse executable code are independent of the language used. WCET tools that analyse source code or intermediate code (e.g., SWEET) are dependent of the availability of the source code, and thus also of the language. Not all languages are handled by existing WCET tools. Typically, C and Ada code is explicitly supported.

Which compiler is used? WCET tools that analyse executable code (e.g., aiT, Bound-T, and RapiTime) are in principle independent of the compiler used. However, decoding of executable code generated by some compilers have better support than others. Often, debug information for the code is used, which means that there is a dependency to the compiler.

WCET tools that analyse source code or intermediate code (e.g., SWEET) are dependent of choice of the compiler. The WCET estimates are valid for the target system only if the same compiler is used both by the WCET analysis and for code generation for the target system.

4.2 Usability Aspects

The three main methodologies are quite different; therefore the discussion below will be separated into three subsections, one for each methodology.

Usability aspects of measurements. There are different approaches for WCET estimation using measurements. Typically, you choose one of the following alternatives:

1. Execute the program on the target hardware with worst-case input and measure the execution time. The problem here is to find the worst-case input. This is not trivial for most programs, since it is often hard to force the program to take a certain path by selecting correct inputs. There are some ideas how to extend the test data domain to get closer to the worst case input, see, e.g., Kirner [15].
2. If the worst-case input is not known, execute the program with all inputs, measure the execution time for each test run, and finally, select the longest time. This is, for most programs, not possible due to combinatorial explosion.
3. An alternative can be to run all paths through the program and select the longest time. This may mean fewer runs than in the last case, since one path typically corresponds to many inputs. However, the number of possible paths for most programs (containing loops and selections) is normally enormous. Also, if execution times are dependent on the inputs, this may not work.
4. A common replacement of the strategies above is to execute the program many times with different inputs, and then select the longest time. However, this is inherently unsafe, since there is no guarantee that the longest path has been executed. It is often very hard to find the real WCET, especially for complex code and/or systems with complex hardware features.

Beside the basic safety problem with measurements, mentioned above, there are some other usability problems that should be mentioned.

- The actual hardware (or a cycle-accurate simulator of the processor) must be available, and the system must be correctly set up.
- All code to be executed must be available.
- Often some initial worst-case configuration have to be made before the measurements, e.g., to guarantee that the cache is empty.
- Disturbances, like interrupts, during measurements are not allowed and must be switched off (or, at least, they have to be accounted for).
- If measurements have probe effects, they do not give the same timing behaviour as the original code.
- It is possible to identify the executed path when hardware like logical analyzers and in-circuit emulators are used. However, a lot of work is typically required. It is therefore very hard to exclude certain paths, like error handling, from measurements. This often requires deep knowledge of the code.

Usability aspects of static WCET tools. When using static tools, a number of things must be done before you start the actual analysis. They are mainly concerned with setting up system info, like processor, clock rate etc. Also, sometimes you will have to supply information about which

code to analyse, address information, etc. If the analysis tool requires the source code, a compilation step is typically required before analysis to ensure agreement between source code and executable code.

One strategy can be to *first* run the WCET tool with no loop bounds. This first attempt does automatic calculation of (some) loop bounds and (some) infeasible paths (different amount of automatic calculation for different tools) and points out the loop bounds that could not be found. This can be due to mainly two reasons, either the loop bound is input-dependent, or the loop is so complex that the loop bound calculation fails.

As a *second* step, solve the problem with the missing loop bounds. If the loop bound is input-dependent, provide input limitations manually using annotations. If a loop is too complex, the loop bound can be given manually, or consider re-writing the code. Now, WCET estimates should be found for the program, maybe with some overestimation.

As a *third* step improve precision by reducing overestimation. If the tool supports input-sensitive flow analysis, limits of input values may be given to lower loop bounds and to find infeasible paths. For other tools, manual annotations may be provided to set limits of register or variable values to eliminate infeasible paths. Manual annotations can also be used to directly exclude infeasible paths or other paths, like error handling, from the analysis. Some tools support low-level annotations that enhance precision by, e.g., giving addresses of memory accesses. This step often requires a lot of work for a complex program, and a deep knowledge of the code.

An important question is on what level manual annotations should be given (source code, intermediate code or executable code level). Different tools allow different levels, e.g., aiT gives you a choice of source code or executable code level, Bound-T uses a simplified way to handle executable code level, and SWEET uses intermediate code code level. As a general rule, source code level annotations are most convenient to give for the programmer (assuming this code is available). Also, source code annotations do not have to be changed after a re-compilation.

Usability aspects of hybrid WCET tools. Since the success of the hybrid tool and the safety of the WCET estimate is based on measurements, basically the same comments as for measurements are valid for these tools. The key issue how to find inputs (test cases) to give the best coverage. Obviously, full path coverage is a, most often, infeasible goal. RapiTime contains functions to find non-covered code, which can help. There are also coverage measures which can be used, which are better than code coverage, like Modified Condition/Decision Coverage (MC/DC). One practical approach could be to start from the test inputs that probably exist for the program to be analysed, and then to extend the input set.

5 Conclusions, Discussion and Future Work

WCET tools are entering a phase where they are more commonly used in industry. However, there is yet no ”one-click” solution; rather, a lot of preparation has to be made before a safe and tight WCET can be found. Research and tool development is necessary to automate this process. A crucial issue for static analysis is to develop powerful and scalable flow analysis methods able to calculate most loop bounds and infeasible paths automatically. For hybrid methods, better methods are required to generate test data.

WCET tools not only calculate WCET estimates. There are a number of other aspects of the tools that are important and make them valuable tools for the programmer. Graphical presentation of programs are very useful, and helps showing the connection between the source code and executable code. By showing WCET of parts of the program, the identification of worst case path and bottlenecks can be simplified. Static tools can help reveal program errors, like dead code and infinite loops. Hybrid tools can enhance the set of test cases. When WCET tools become a natural and integrated part of the development environment, they can assist the programmer in developing reliable software.

Since different WCET analysis methods have different advantages and disadvantages, they should be combined to support each other. For example, rough estimates using measurements or hybrid methods can be used at the beginning of the timing analysis phase, while safe and rigorous WCET values found by static tools are necessary towards the end. Measurements and statically found WCET values limits the real WCET from below and above, and thus we can both know the possible interval of the real WCET, and the maximal overestimation made by the statically found WCET value. Also, knowledge about the (possibly overestimated) worst-case path found by static analysis could be used during measurements to search for the real worst-case path. Static and hybrid tools can exchange information that may enhance their respective analyses.

The newly started EU project ”Integrating European Timing Analysis Technology” (with project acronym ALL-TIMES) [2] within the EU Frame Program 7 contains themes for research including interoperation, scalability, automation, and integration of timing analysis tools. ALL-TIMES will thus consider many of the issues discussed in this paper.

References

[1] AbsInt. aiT tool homepage, 2008. `www.absint.com/ait`.

[2] Homepage for the ALL-TIMES project, 2008. `www.all-times.org`.

[3] HJ. Bang, TH. Kim, and SD. Cha. An iterative refinement framework for tighter worst-case execution time

calculation. In *The 10th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC2007)*, volume 00, pages 365–372, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[4] The Chronos WCET analysis tool homepage, 2006. www.comp.nus.edu.sg/∼rpembed/chronos.

[5] Ola Eriksson. Evaluation of static time analysis for CC systems. Master's thesis, Mälardalen University, Sweden, August 2005. 63 pages, www.mrtc.mdh.se/publications/0978.pdf.

[6] Andreas Ermedahl and Jakob Engblom. Execution time analysis for embedded real-time systems. In Insup Lee, Jospeh Y-T. Leung, and Sang H. Son, editors, *Handbook of Real-Time and Embedded Systems*, pages 35.1 – 35.17. CRC Press, 2007.

[7] 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 *Proc. 1^{st} International Workshop on Embedded Systems, (EMSOFT2000), LNCS 2211*, Oct 2001.

[8] J. Gustafsson. The worst case execution time tool challenge 2006. In *Proc. 2^{nd} International Symposium on Leveraging Applications of Formal Methods (ISOLA'06)*, November 2006.

[9] Jan Gustafsson and Andreas Ermedahl. Experiences from applying WCET analysis in industrial settings. In *The 10th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC2007)*, Santorini Island, Greece, May 2007.

[10] Jan Gustafsson, Andreas Ermedahl, Christer Sandberg, and Björn Lisper. Automatic derivation of loop bounds and infeasible paths for WCET analysis using abstract execution. In *Proc. 27^{th} IEEE Real-Time Systems Symposium (RTSS'06)*, December 2006.

[11] C. Healy, M. Sjödin, V. Rustagi, D. Whalley, and R. van Engelen. Supporting timing analysis by automatic bounding of loop iterations. *Journal of Real-Time Systems*, 18(2-3):129–156, May 2000.

[12] Homepage for the Heptane WCET analysis tool, 2006. www.irisa.fr/aces/work/heptane-demo.

[13] N. Holsti, T. Långbacka, and S. Saarinen. Worst-case execution-time analysis for digital signal processors. In *Proc. EUSIPCO 2000 Conference (X European Signal Processing Conference)*, 2000.

[14] Niklas Holsti, T. Långbacka, and S. Saarinen. Using a worst-case execution-time tool for real-time verification of the DEBIE software. In *Proc. DASIA 2000 Conference (Data Systems in Aerospace 2000, ESA SP-457)*, September 2000.

[15] Raimund Kirner, Ingomar Wenzel, Bernhard Rieder, and Peter Puschner. Using measurements as a complement to static worst-case execution time analysis. In *Intelligent Systems at the Service of Mankind*, volume 2. UBooks Verlag, Dec. 2005.

[16] Pascal Montag, Steffen Goerzig, and Paul Levi. Challenges of timing verification tools in the automotive domain. In *Proc. 2^{nd} International Symposium on Leveraging Applications of Formal Methods (ISOLA'06)*, Paphos, Cyprus, November 2006.

[17] OTAWA homepage, 2007. http://www.irit.fr/recherches/ARCHI/MARCH/rubrique.php3?id_rubrique=28.

[18] PapaBench homepage, 2007. http://www.irit.fr/recherches/ARCHI/MARCH/rubrique.php3?id_rubrique=97.

[19] Peter Puschner. The single-path approach towards WCET-analysable software. In *Proc. IEEE International Conference on Industrial Technology*, pages 699–704, Dec. 2003.

[20] RapiTime WCET tool homepage, 2006. www.rapitasystems.com.

[21] Mälardalen WCET research group. Mälardalen WCET benchmarks homepage, 2006. http://www.mrtc.mdh.se/projects/wcet/benchmarks.html.

[22] M. Rodriguez, N. Silva, J. Esteves, L. Henriques, D. Costa, N. Holsti, and K. Hjortnaes. Challenges in calculating the WCET of a complex on-board satellite application. In *Proc. 3^{rd} International Workshop on Worst-Case Execution Time Analysis, (WCET'2003)*, 2003.

[23] Jan Staschulat and Rolf Ernst. Worst case timing analysis of input dependent data cache behavior. *ecrts*, 0:227–236, 2006.

[24] L. Tan. The worst case execution time tool challenge 2006: The external test. In *Proc. 2^{nd} International Symposium on Leveraging Applications of Formal Methods (ISOLA'06)*, November 2006.

[25] S. Thesing, J. Souyris, R. Heckmann, F. Randimbivololona, M. Langenbach, R. Wilhelm, and C. Ferdinand. An abstract interpretation-based timing validation of hard real-time avionics software. In *Proc. of the IEEE Int. Conf. on Dependable Systems and Networks (DSN-2003)*, June 2003.

[26] Tidorum. Bound-T tool homepage, 2006. www.tidorum.fi/bound-t.

[27] Ingomar Wenzel, Bernhard Rieder, Raimund Kirner, and Peter Puschner. Automatic timing model generation by CFG partitioning and model checking. In *Proc. Conference on Design, Automation, and Test in Europe (DATE)*, Mar. 2005.

[28] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution time problem — overview of methods and survey of tools. *Accepted for publication in ACM Transactions on Programming Languages and Systems*, 2008.

[29] Yina Zhang. Evaluation of methods for dynamic time analysis for CC-systems AB. Master's thesis, Mälardalen University, August 2005. 72 pages, www.mrtc.mdh.se/publications/0977.pdf.