

# A Simple and Effective Fully Automatic Worst-Case Execution Time Analysis for Model-Based Application Development \*

Raimund Kirner<sup>1</sup>, Peter Puschner<sup>1</sup>

<sup>1</sup>Institut für Technische Informatik  
Technische Universität Wien  
A-1040 Wien, Austria  
{raimund,peter}@vmars.tuwien.ac.at

**Abstract** — *Embedded systems typically have to fulfill certain real-time constraints. Worst-case execution time analysis (WCET) has to be done to reason about the timing behaviour of those systems. Current techniques for static WCET analysis tools are not mature enough to be used for modern processors in an industrial-strength environment. In this paper we present a WCET analysis approach that allows to calculate the WCET of modern processors automatically. The approach exploits additional knowledge about the code structure in application specific domains in combination with an hybrid analysis method. The hierarchical application development environment MATLAB/Simulink has been selected as an example for such application specific domains for WCET analysis.*

## 1 Introduction

The knowledge of the worst-case execution time (WCET) is mandatory for the design of real-time systems [1]. Since about more than one and a half decades, research in WCET analysis has been done to support the industry with concepts to enable the development of WCET analysis tools. Still there is hardly any impact to the industrial practice of timing analysis [2].

Due to the missing of industrial-strength WCET analysis tools, typical methods for timing analysis in industrial environments are the manual counting of instruction cycle times or performing sample runtime measurements. Both methods have their obvious drawbacks. Manual counting of instruction cycle times becomes infeasible for the complete analysis of real-size programs. The danger of performing sample runtime measurements is the probability that the most critical execution paths were not covered by the sample input data.

---

\*This work has been supported by the FIT-IT research project “Model-Based Development of distributed Embedded Control System (MoDECS-d)”.

On the other side, the problem with static WCET analysis approaches in general is that undecidability in combination with modern processors having pipelines and caches makes it impossible to derive safe and tight upper bounds for the WCET. The current practice to overcome these limitations is to use code annotations that will guide the WCET analysis tool to find the longest execution path through the program.

In this paper we present an intelligent solution to exploit the advantages of the two concepts, the static WCET analysis and the runtime measurements. We describe an industrial strength WCET analysis framework that overcomes the limitations of existing WCET analysis methods. The presented WCET analysis approach is intended for soft real-time systems and calculates the WCET of programs fully automatically. To achieve this, on the one side we specialize the analysis to certain applications contexts. We concentrate the analysis to code generated from MATLAB/Simulink<sup>1</sup> by exploiting knowledge about the structure of the code. In contrast to our previous work [3, 4], we do not have to change the code generator of MATLAB/Simulink for the WCET analysis method presented in this paper. We use a hybrid approach that combines static analysis with runtime measurements. Basically, we use a C-source-level machine that is calibrated against the real target processor. Some experiences with a calibration of a performance model for the PowerPC have been reported by Black and Shen in [5].

The paper is structured as follows: Section 2 gives a discussion about demands from industry for the use of WCET analysis tools. Section 3 presents our new WCET analysis concept. A discussion of the presented WCET analysis framework is given in section 4. Finally, section 5 concludes this paper.

## 2 Requirements for an Industrial-Strength WCET Analysis Tool

Before describing our new WCET analysis method, we give a motivation for its development by describing the industrial needs on a WCET analysis method. Previously proposed WCET analysis methods often only demonstrate certain analysis capabilities without showing their applicability in an industrial environment. To be more precise, the following list gives demands for a WCET analysis tool raised by peoples working in industry. This list also contains aspects regarding the use of *hierarchical application development environments* (HADE), as they are increasingly used in industrial software development.

1. The tool must work with minimal user interaction. In particular, it cannot be expected that users of the tool provide manual code annotations about possible and impossible execution paths of the code. The tool must be able to extract this information by analyzing the code generated by the code generator of the HADE.
2. The method must integrate into the development tool chain of customers without modification of tools from the tool chain (e.g., components of HADE, code generator, C compiler).

It may be possible to use the tool chain in a restricted manner to enable the application of a certain WCET analysis method. For example, the available application

---

<sup>1</sup><http://www.mathworks.com>

development features of a HADE tool may be restricted or certain compiler optimizations may be deactivated.

3. The method must be easily adaptable to new releases of software components of the tool chain. Expensive adaptations of the WCET method to new releases of software components have to be avoided.

The situation that a development tool of the tool chain explicitly supports a specific WCET analysis methods is currently very rare. For example, it can be possible that a compiler provides certain support to perform WCET analysis [6, 7]. But such tools are typically in a prototype phase without commercial support.

Therefore, the best current strategy for developing a WCET analysis tool is to cope with existing COTS software development tools.

4. The WCET analysis method must be easy to retarget to different hardware settings, i.e., the implementation or configuration effort must be low enough for an economic useability of the WCET analysis method. Depending on the concrete WCET analysis method, there are in principal two different possibilities for retargetability. First, it can be required to order further implementation effort from the WCET tool provider. Second, it may be possible that the tool is flexible enough so that the customer can adopt the tool by himself. The latter approach is applicable for adequate measurement based WCET analysis methods.

The adaption of a WCET analysis method to new hardware configurations can be kept easy when the WCET analysis method is based on measurements on the real hardware. Because in this case the WCET analysis method does not have to provide a so-called *exec-time model* which describes the execution times for given code sequences. Such an exec-time model is substituted by measurements on the real target hardware. There exist also measurement-based WCET analysis approaches that use a hardware simulator instead of measurements on the real hardware [8, 9]. Such approaches rely on the existence of an accurate hardware simulator which is often not available.

In the following section a new WCET analysis method is presented that is able to fulfill the given above requirements from industry. As an example for a HADE tool, we use MATLAB/Simulink as it is applied in industry increasingly as an abstract application development platform. The WCET analysis for applications developed with HADE tools becomes easier because there is additional information available about the structure of the generated code.

### 3 WCET Analysis Using an Abstract Execution Model

This section introduces a hybrid WCET analysis technique that combines advantages of both the static WCET analysis paradigm and the dynamic WCET analysis paradigm. The dynamic part is used to construct by measurements an exec-time model that is used within the part of static WCET analysis.

#### 3.1 The C-Source-Level Machine

To obtain accurate results, WCET analysis is typically performed at object code level. That means that a WCET analysis uses a machine model of the target processor to calculate the WCET. Performing WCET analysis at object code level can be the base for

a tight and accurate calculation. However, this approach has several difficulties. Static WCET analysis methods that use flow information from the source code have to consider the effects from an optimizing compiler and have to model the behaviour of the target processor. Dynamic WCET analysis methods have to perform exhaustive measurements for the timing analysis of each program.

The solution we present in this paper is to use a hybrid WCET analysis technique. We use an abstract target machine which we call *C-source-level machine* because our analysis is based on ANSI C code fragments. But the concepts can be easily applied to other imperative programming languages. Non-imperative programming languages are less suited for this approach since it would be more difficult to map the program's control flow of such programming languages to the executions performed by the target processor. Therefore, performing WCET analysis on non-imperative programming languages introduces additional pessimism compared to analysis of imperative programming languages.

Performing WCET analysis at source code level like C is not completely new. Park and Shaw applied their WCET calculation based on *timing schema* to a subset of the programming language C [10]. They assumed a fixed mapping from source language constructs to assembly code statements, which prevents the consideration of code optimizations performed by a compiler. To summarize, they build a static exec-time model for their C-source-level machine. As a consequence, their approach results into inherent overestimations of the real WCET, because they cannot consider performance improvements due to code optimizations performed by the compiler.

In contrast to the above statically constructed exec-time model, we construct an exec-time model by calibration based on measurements on the real target hardware. Further, we choose a different granularity level than simple source statements. This is feasible since we consider MATLAB/Simulink as a HADE tool where it is possible to identify fractions of the C code that belong to a certain building block of the HADE tool. That means that an instruction (denoted as *analysis unit*) of our C-source-level machine consists of multiple C source statements. The exact granularity of an analysis unit has to be derived by experiments. A first attempt would be to assign *analysis units* to single building blocks of the HADE tool.

### 3.2 Calibration of the Exec-Time Model

To construct an accurate exec-time model for our C-source-level machine, we use a two-phase calibration procedure. These two phases are shown in figure 1. The invocation of tools is represented by rectangular boxes. Data like program representations are represented by rounded boxes. The tools that are specific for this calibration procedure are drawn as gray boxes.

For calibration phase I it is necessary to specify the set of relevant building blocks from the HADE tool (in our specific case MATLAB/Simulink block sets). The *Unit Selection* tool selects an *analysis unit* that is converted by the *Model Creation* tool into a small application model. These models are processed by the standard tool chain consisting of a code generator for the HADE tool and the compiler to generate object code for the target hardware. This object code is analyzed by the *Runtime Measurement* tool to obtain its execution time. These measurements can be performed and stored for different tool chain configurations, e.g., different compiler optimizations. To improve the accuracy of the

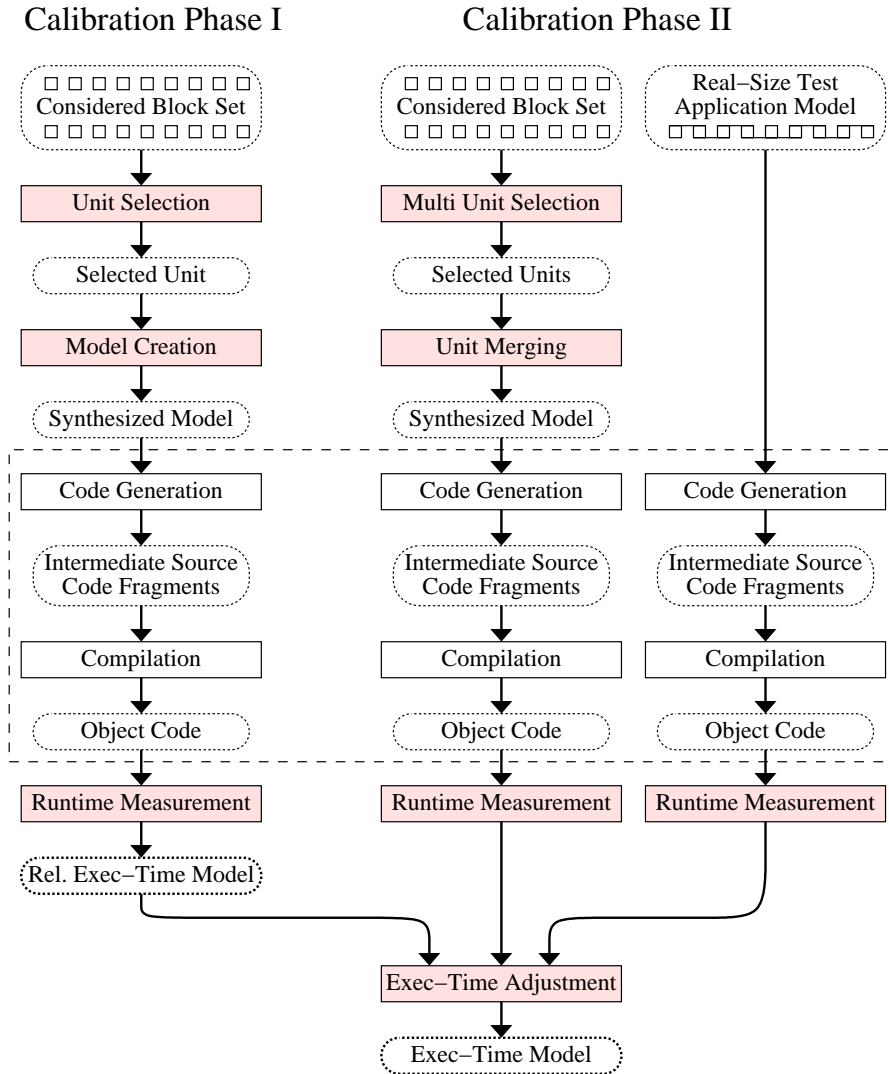


Figure 1: Calibration Phases of the WCET Tool

results, these runtime measurements for the generated code from single *analysis units* are only used to record the relative execution times of them. These relative execution times are stored in a database called *Rel. Exec-Time Model*.

The calibration phase II uses the result from calibration phase I together with measurements on combinations of analysis units and real-size application models to construct the exec-time model for the C-source-level machine. This calibration phase measures the execution time of combinations of analysis units and also real-size models. The tools *Multi Unit Selection* and *Unit Merging* are used to construct application models for combinations of analysis units. Finally, the stage *Exec-Time Adjustment* uses the result from the runtime measurements together with the database *Rel. Exec-Time Model* from calibration phase I to calculate the *exec-time model* which describes the execution time for single analysis units.

### 3.3 The WCET Analysis Tool

Based on the exec-time model which has been derived by measurements as described in section 3.2, we use a static WCET analysis approach to calculate the execution time of the C source code generated from the HADE application model.

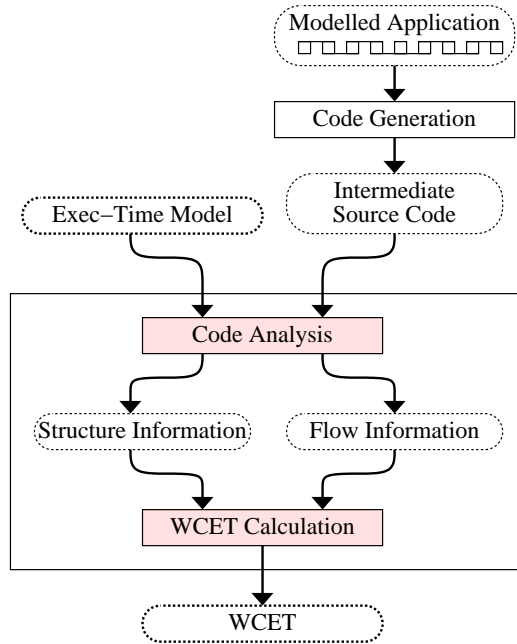


Figure 2: Components of the WCET Analysis Tool

Figure 2 shows the basic steps to calculate the WCET. The stage *Code Analysis* is a pre-processing step for the final WCET calculation. The *Code Analysis* stage structures the intermediate source code into analysis units and applies the exec-time model on them. Depending on the code structure the code analysis may also extract further control flow information that can guide the WCET analysis stage to find the longest execution path through the code. Typical flow information are bounds for the iteration counts of loops, or (in)feasible paths. Infeasible paths are control-flow paths that can be never taken during program execution. Infeasible paths can be identified by examining the semantics of program code or also by considering knowledge about the possible values of a program’s input parameters.

The bound for the WCET is calculated in the stage *WCET Calculation*. Our WCET analysis framework does not rely on a specific WCET calculation method. The *implicit path enumeration technique* (IPET) [11] has been chosen because it has several advantages compared to other methods like tree-based or path-based WCET calculation. IPET is easy to implement because it maps the WCET calculation to an integer linear programming problem that can be solved with standard constraint solvers. The resulting integer linear programming problems are sufficiently small to use a free available solver like *lp\_solve*<sup>2</sup> to calculate the WCET.

<sup>2</sup>available by anonymous ftp from [ftp://ftp.es.ele.tue.nl/pub/lp\\_solve](ftp://ftp.es.ele.tue.nl/pub/lp_solve)

A fundamental limitation for the application of WCET analysis is the question of decidability. The so-called *Halting Problem* implies that it is in general not possible to calculate the WCET of a program. To overcome this limitation, *flow facts* have to be provided manually. For the calculation of the WCET flow facts are required that contain at least information about the iteration bounds of loops. For example, programs written in a programming language like ANSI C can require the provision of additional flow facts to enable the calculation of WCET bounds.

In our approach we avoid the necessity of providing flow facts by using a HADE for program development. A HADE represents an application specific domain of program development. The program development with a HADE can be more intuitive but imposes restrictions on the shape of the resulting code. For example, the code generated from a MATLAB/Simulink simulation model does not contain input-data dependent loops. The loops in the generated code are created by the code generator to operate with multi-dimensional data types like vectors or matrices. The special nature of MATLAB/Simulink is that it enforces a data-flow oriented application development. Circles in the data-flow model do not imply that the generated code contains loops. The circles in the data-flow model are resolved in the generated code by repeated calls to the calculation function. Specializing the WCET analysis to code generated from a HADE like MATLAB/Simulink allows the *Code Analysis* stage of the WCET analysis framework to extract all required flow information automatically without user interaction. Describing it more abstract, by specializing the WCET analysis to an application specific domain it was possible to make fully automatic WCET analysis feasible.

## 4 Discussion of the Approach

This section gives a discussion about properties of the presented WCET analysis framework and raises open questions for further research.

### 4.1 Construction of the Exec-Time Model

The WCET analysis tool presented in section 3.3 is similar to traditional static WCET analysis approaches in that it uses an exec-time model to obtain the execution time of instruction sequences. The main difference to traditional approaches is that this exec-time model is constructed at an hardware-independent abstraction level based on the source code representation of the application.

The construction of the exec-time model as described in section 3.2 can be easily adopted to new hardware configurations as the only hardware-dependent component is the *runtime measurement* tool. Applying the *runtime measurement* to a new hardware configuration only requires to exchange COTS components within the measurement setup. Therefore, this WCET analysis approach based on a calibrated exec-time model can be retargeted to a new hardware configuration with significant less effort as it is required on static WCET analysis frameworks.

A further advantage of the approach is that the calibration of the exec-time model is a fully automatic task that could be also performed by the user of the WCET analysis tool itself. Each time the hardware setting is changed, the calibration can be redone to calculate the WCET bound based on an up-to-date exec-time model.

## 4.2 Accuracy of the Calculated WCET Bound

The approach is intended for soft real-time applications where deadline misses do not result into catastrophic events. As small underestimations of the WCET can be tolerated, it allows the construction of a more precise WCET analysis tool with reduced overestimations.

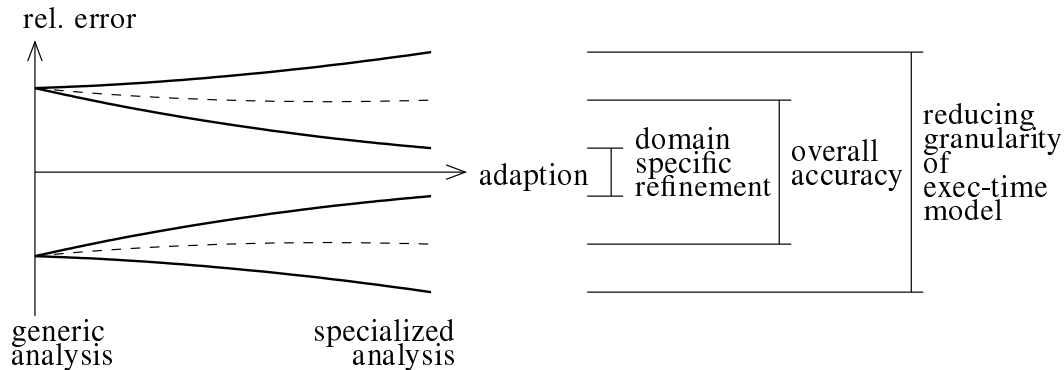


Figure 3: Tendency of Analysis Errors by Adapting the WCET Analysis Technique

A currently open topic is the achievable accuracy of the presented WCET analysis framework. A traditional WCET analysis framework typically analyses object code by using a timing model that has a granularity of a single machine command. With the help of code annotations, a traditional WCET analysis framework can calculate a bound of the WCET. The WCET analysis framework presented in this paper has special properties that makes it hard to compare its accuracy with that of traditional frameworks. For clarification, the relative WCET bound calculation error of both concepts is shown in figure 3. The traditional WCET analysis framework is denoted as *generic analysis* and the approach presented in this paper is denoted as *specialized analysis*. Starting from the *generic analysis*, the relative error of the calculated WCET bound is within a certain uncertainty interval. For the construction of the *specialized analysis* we have applied two strategies:

**domain specific refinement:** By specializing the WCET analysis on applications developed by a HADE like MATLAB/Simulink it is possible to calculate the longest execution trace of a code more precisely even without code annotations. As shown in figure 3, the relative error is reduced by this strategy.

**reducing granularity of exec-time model:** The exec-time model traditionally has a granularity of a single machine command. As described in section 3.1, the granularity of the WCET analysis framework described in this paper is one *analysis unit*. An *analysis unit* consists of several C source statements, hence the granularity of the exec-time model is coarser than that of *generic analysis*. As shown in figure 3, the mean value and the uncertainty interval of the relative error is increased by using a coarser exec-time model.

The impacts of these two strategies on the overall relative error of the WCET analysis framework are contrary. The *overall accuracy* for the application of both strategies is shown in figure 3 as an intermediate value from that of these two strategies. Whether the overall accuracy of the *specialized analysis* or the *generic analysis* is better depends on the overall impact of both strategies. Future research is required to identify a useful granularity for an *analysis unit*.

## 5 Summary and Conclusion

In this paper we described a novel WCET analysis approach that is well-suited for the use in industrial-strength environments. The approach is based on two important concepts to perform fully automatic WCET analysis, while requiring only simple adaptations of the framework in case of new target hardware configurations. First, we use a hierarchical application development environment like MATLAB/Simulink which provides several knowledge about the structure of the generated code. Second, we perform WCET analysis at an hardware-independent level by using an exec-time model that is calibrated with the target hardware in two phases for improved accuracy.

The presented WCET analysis framework can be adopted easily to new hardware configurations as it is only required to rerun the automatic calibration facility for the new configuration of the target hardware.

The specialization of the WCET analysis to the application specific domain of programs developed with hierarchical application development environments like MATLAB/Simulink make it feasible to calculate a WCET bound without additional code annotations that describe the possible control flow. Therefore, a WCET bound of code generated from MATLAB/Simulink can be calculated fully automatically. Furthermore, the usage of an exec-time model that can be derived automatically by calibration measurements from a given hardware configuration makes it easy to retarget the WCET analysis framework to a new hardware configuration. The retargeting to a new hardware configuration can be done by the user itself as it is only required to trigger the measurements to recalibrate the exec-time model.

The determination of a useful granularity of the *analysis unit* is left to future research.

## References

- [1] Peter Puschner and Alan Burns. Writing Temporally Predictable Code. In *Proc. 7th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 85–91, Jan. 2002.
- [2] Peter Puschner. Is worst-case execution-time analysis a non-problem? – towards new software and hardware architectures. In *Proc. 2nd Euromicro International Workshop on WCET Analysis*, Technical Report, York YO10 5DD, United Kingdom, Jun. 2002. Department of Computer Science, University of York.
- [3] Raimund Kirner, Roland Lang, Peter Puschner, and Christopher Temple. Integrating WCET Analysis into a Matlab/Simulink Simulation Model. In *Proc. 16th IFAC Workshop on Distributed Computer Control Systems*, Sydney, Australia, Nov. 2000. School of Computer Science and Engineering, UNSW.
- [4] Raimund Kirner, Roland Lang, Gerald Freiberger, and Peter Puschner. Fully automatic worst-case execution time analysis for matlab/simulink models. In *Proc. 14th Euromicro Conference on Real-Time Systems*, pages 31–40, Vienna, Austria, Jun. 2002. Vienna University of Technology, IEEE.

- [5] Bryan Black and John P. Shen. Calibration of microprocessor performance models. *Computer*, 31(5):59–65, May 1998.
- [6] Raimund Kirner and Peter Puschner. Transformation of Path Information for WCET Analysis during Compilation. In *Proc. 13th IEEE Euromicro Conference on Real-Time Systems*, pages 29–36, Delft, The Netherlands, Jun. 2001. Technical University of Delft.
- [7] Raimund Kirner and Peter Puschner. Timing analysis of optimised code. In *Proc. 8th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2003)*, Guadalajara, Mexico, January 2003.
- [8] Jakob Engblom and Andreas Ermedahl. Pipeline timing analysis using a trace-driven simulator. In *Proc. 6th International Conference on Real-Time Computing Systems and Applications*, Hong Kong, Dec. 1999.
- [9] Thomas Lundqvist and Per Stenström. Integrating path and timing analysis using instruction-level simulation techniques. In *Proc. ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 1–15, Jun. 1998.
- [10] Chang Y. Park. Predicting Program Execution Times by Analyzing Static and Dynamic Program Paths. *Real-Time Systems*, 5(1):31–62, 1993.
- [11] Peter Puschner and Anton V. Schedl. Computing Maximum Task Execution Times – A Graph-Based Approach. *The Journal of Real-Time Systems*, 13:67–91, 1997.