Time-Predictable Computing*

Raimund Kirner[1] and Peter Puschner[2]

[1] Department of Computer Science
University of Hertfordshire, United Kingdom
r.kirner@herts.ac.uk
[2] Institute of Computer Engineering
Vienna University of Technology, Austria
peter@vmars.tuwien.ac.at

Abstract. Real-time systems need to be *time-predictable* in order to prove the timeliness of all their time-critical responses. While this is a well-known fact, recent efforts of the community on concretizing the predictability of task timing have shown that there is no common agreement about what the term *time-predictability* exactly means. In this paper we propose a universal definition of time-predictability that combines the essence of different discussions about this term. This definition is then instantiated to concrete types of time-predictability, like worst-case execution time (WCET) predictability. Finally, we introduce the concept of a timing barrier as a mechanism for constructing time-predictable systems.

1 Introduction

Research on real-time computing is an established field with several decades of research. Time-predictability is exactly one of the preconditions necessary to verify the correct operation of a real-time system. However, except for some recent efforts, there was no visible attempt to precisely define what time-predictability, characterizing the temporal behavior of tasks, exactly means. This might be explained by the fact that traditional research on real-time computing was based on design choices where the available components were (implicitly) sufficiently time-predictable. However, with the increasing performance gap between processing speed and memory access times, the commercial computer components available on the market have become more complicated, as they introduced peak-performance enhancing features like pipelines and caches. With the currently observed stall on maximal performance of single-core processors, the situation again has become more challenging for timing analysis – nowadays we observe a strong move towards the use of multi-core systems in which the competition for memories that are shared among the processor cores negatively affects predictability.

* The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7,2008-2011] under grant agreement no 214373 (ArtistDesign, http://www.artist-embedded.org/) and the IST FP-7 research project Asynchronous and Dynamic Virtualization through performance ANalysis to support Concurrency Engineering (ADVANCE).

S.L. Min et al. (Eds.): SEUS 2010, LNCS 6399, pp. 23–34, 2010.

Analysis of the worst-case execution time (WCET) [1] for systems built from conventional components is relatively complicated [2,3]. On one hand it is very difficult to re-engineer the behavior of a processor in order to construct a model that reflects the timing of all operations of the processor correctly. On the other hand high computational effort is needed to calculate the WCET bound.

To react on the current hardware trend towards computer systems with increasingly complex temporal behavior, the timing analysis community started to work on simpler hardware components for which WCET analysis can be done with reasonable effort. To systematize this effort, it is advantageous to first work out a common understanding of *time-predictability*. This seems to be important as we observed that the term time-predictability has been used with different meanings in the timing-analysis community so far. To give an example, Thiele et al. [4] define time-predictability as the pessimism of WCET analysis and BCET analysis, while Grund defines time-predictability as the relation between BCET and WCET [5].

In this paper we contribute a new view on time-predictability of real-time tasks. The definition of the term time-predictability is based on a timing model used for WCET analysis. Besides time-predictability, we also discuss the efforts needed to construct such a model, as well as the necessary analysis efforts and the stability of the obtained timing behavior.

2 Worst-Case Execution Time Analysis

One of the main reasons for writing this paper is that we observe that there are different, even divergent, understandings of what *predictable task timing* is meant to stand for (see above).

Task timing analysis and WCET analysis are just a part of the software development process for dependable hard real-time systems. The whole software development process is hierarchical, in order to make the design and implementation of complex applications simple.

The higher levels of the development process deal with the decomposition of the application software into groups of tasks and finally single tasks, they assign tasks to the hardware components of the computer system, and they construct and check schedules that fulfill the communication and synchronization requirements of the tasks in the application. The task level of software development has to make sure that the tasks deliver the correct functionality and meet their execution-time constraints. To achieve the latter, the timing of tasks, in particular their WCET, should be easy to assess, i.e., task timing should be in some way forseeable.

The task interface separates the task level from the higher levels of software development. It characterizes the types, possible values and the meaning of the inputs and outputs, and the semantics and execution time of the task. To support the hierarchical development, a task system needs to be (nearly) composable, i.e., the task interfaces have to be small and the interactions of the tasks via the interface should be weak [6] – strong interactions or dependencies via an interface would either necessitate a complex task-wise analysis that took these these dependencies into account or, alternatively,

would yield highly pessimistic results when combining the results of the analyses of the individual tasks.

In accordance to the above-mentioned ideas the tasks subject to WCET analysis are generally simple tasks (see [7] for the description of the simple-task model). Simple tasks have a plain *input – processing – output* structure, where all inputs are available (and read) at the beginning, outputs are available (and written) at the end, and the central part of the task is free of any I/O, task synchronization, waiting, or delays.

In the following we understand a task as an object that is defined by its code and the hardware on which the code is expected to run. We assume that a task executes without any disturbances by other tasks or system activities. Task interactions happen via the tasks data interface (consisting of inputs from the environment and the values of its state variables at the beginning of its execution), and via the state of the hardware at the task start time which might differ between different executions. Note that we assume that the state of a task – the set of state variables that keep their values between successive executions – is part of its input. Stateful tasks are modeled by tasks that send the values of their state variables at the end of each execution and read them in again at the beginning of their next execution. Making the state of a task part of its input has the advantage that task timing can be modeled locally, without the need to model how the task state might develop over a number of task instantiations.

Possible Understandings of Temporal Predictability

The term *temporal predictability* respectively *time predictability* can be understood in different ways, depending on which questions about the temporal behavior of tasks one wants to answer. The different possible understandings of the term can be grouped into three categories.

- Predictability as a statement describing the properties of the phenomenon execution time of a task, e.g., the possible range or possible occurrence patterns of execution times of the task.
- Predictability as a statement about the properties of the process of modeling the timing behavior of a task. Notions of this type might be used to answer questions about how difficult it is to build or evaluate a timing model for some code running on some target processor, or what the cost of constructing such a model are.
- A combination of the above.

Predictability as Statement about the Phenomenon Execution Time

In this section we present some interpretations of time predictability when this term is meant to characterize properties of the phenomenon execution time of a task, assuming an undisturbed execution of the task and assuming a given task software and target hardware.

- "For given input data and a given hardware state at the time the task starts the execution time of an execution is determined." – predictability as a statement about the deterministic behavior of hardware and software.

- "For a given set of possible inputs and start states there is a minimum execution time (BCET) and a maximum execution time (WCET) that define an interval [BCET, WCET] in which all possible execution times of the task can be found." – predictability as a measure for the variability of execution times. Obviously, a smaller interval yields a smaller expected error for "guesses" about the execution times of executions. Thus the smaller the interval [BCET, WCET] the better the predictability in this understanding. The special case that BCET = WCET yields the highest predictability ranking in this understanding.
- Another interpretation of predictability could aim at describing how the execution time of a task develops over time, i.e., from one execution to the next, etc. For example, think of a periodic tasks whose execution times show the repeated pattern $t1$, $t2$, $t3$. One might consider this as a very predictable execution-time pattern. While this pattern might indeed be considered predictable at a higher level, such execution-time patterns are beyond what we can argue about at the task level. Arguing about execution time patterns requires that we need to know how the inputs of the task change from one task instance to the next. This is however beyond the scope of what can be expected and done at the task level. When considering a task – purely at the task level, i.e, independent of how inputs evolve – arguing about the timing dynamics over time does not make sense.

The following sections of the paper deal with the understandings of timing predictability that also incorporate the properties of the execution-time modeling process, i.e., how close the modeled timing resembles the real execution times, respectively how costly it is to build an execution-time model.

3 Time-Predictability

In the following we present a definition for predictability of the task timing. We build this definition around an abstract meaning of timing, and concretize it then for two special cases, the predictability of the worst-case execution time (WCET) and the best-case execution time (BCET). We start with a discussion of predictability in general.

3.1 Constructability of Timing Model

The construction of an adequate system model for the type of prediction one is interested in can be quite challenging, for which we identify three reasons, as shown in Figure 1:

1. It can be challenging to observe the system behavior in sufficient detail, as certain internal mechanisms might not be accessible by the provided interfaces.
2. Once we get all the relevant details that are necessary to model the timing of a system, the model might become so complex that it is hard to understand the behavior at an adequate abstraction level for the modeling purpose.
3. It has to be ensured that the model reflects the modeling purpose in a safe way, for example, by ensuring that the model provides an overapproximation of the system's possible behavior.

The level of complexity of these three steps are an indicator for the constructability of the model. While the constructability of the timing model of a system is somehow related to the predictability of the timing model, we consider these two aspects separately. Keeping constructability of predictability of the model separate is motivated by the fact that the construction of the hardware part of the model has to be done only once for each processor, while this model can subsequently be used for the prediction of each program running on that hardware.

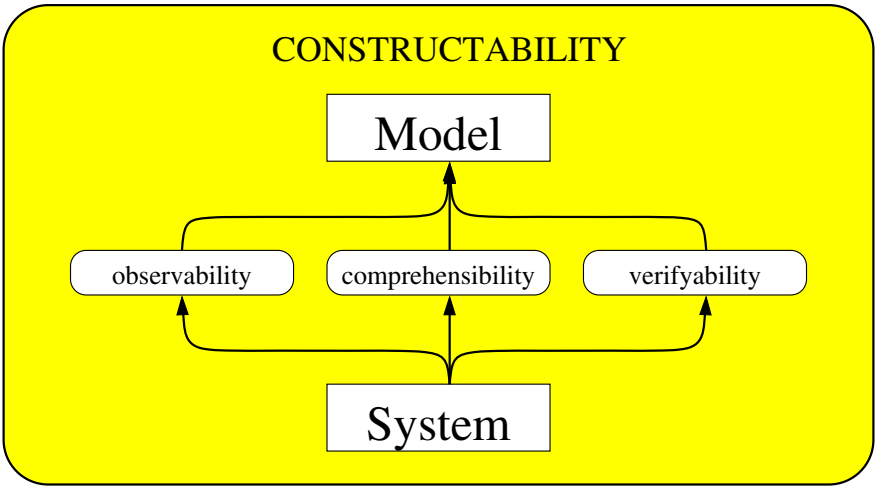

Fig. 1. Constituents for Constructability

3.2 Constituents of Predictability

When we try to predict a system's behavior we do this by analyzing a (simplified) model of the system, i.e., the model is the central object for prediction. Discussing the predictability of a system thus results in a discussion of the system model.

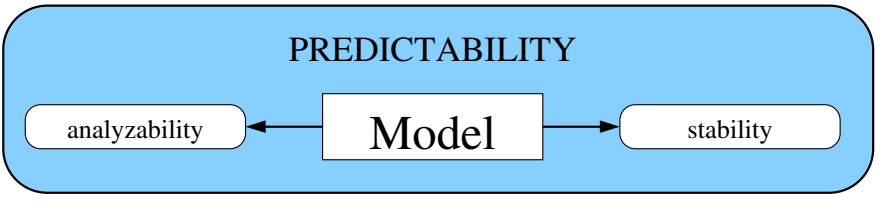

Fig. 2. Constituents for Predictability

As shown in Figure 2, the aspects of the model to be discussed for judging the system's predictability are the *analyzability* and the *stability* of the model.

Analyzability of the model: To make a prediction we have to analyze the model for the properties of interest. The analysis of the model is different from the verification of the model. Even if the correctness of the model can be ensured based on simple construction rules for which the overall correctness can be shown by induction, the analysis of the model to calculate a parameter of interest can still result in a global search problem over the whole model. The degree of analyzability depends on how much effort has to be spent to calculate the investigated property and on how accurate the obtained result is.

Stability of the model: The predictability of the model in general also depends on the degree of variation in the investigated behavior. The fewer open parameters the model has, the more exact a prediction can be made. Furthermore, the smaller the variation in the possible behavior is, the more exact a prediction can be made.

These two aspects in combination determine the level of time predictability.

3.3 Predictability of Task Timing

To describe the task timing, the system model has to contain information about the executed program instructions and the durations of these actions on the concrete hardware platform. Constructing a precise and correct timing model is typically quite challenging. This is mostly because the temporal processor behavior is not very well documented, for example, to hide intellectual properties of the hardware implementation, or simply due to the lack of the construction of a precise model during the processor design phase. It might be also the case that the processor manual simply contains errors on the processor's timing behavior. The missing level of detail in the documentation also reduces the comprehensibility of the timing results observed by experiments.

The analyzability of the timing model depends both on the complexity of the program to be analyzed and on the complexity of the hardware platform. The properties to evaluate are the accuracy obtained by the timing analysis and the computation effort needed to achieve this result. This implies that the availability of an efficient analysis method is a prerequisite for a good time-predictability.

At a first glance, defining time-predictability not only on the properties of the system but also on the properties of the available analysis methods might seem surprising. However, if we look closer and try to discuss time-predictability (or predictability in general) without considering the scope of any calculation technique, we would lose the foundation for any ordering relation between different degrees of time-predictability. What remains would be a boolean decision of whether the requested information is decidable or not (from a theoretical point of view). Therefore, the calculation cost for deriving the information of interest is of importance for a practical definition of time-predictability. An indicator for the computational effort of the analysis is the explicity of information [8]:

> *Explicity of Information:* Given a system model, the *explicity of information* refers to the computational complexity required to infer this information. While a relative explicity ordering of different information typically depends on the underlying calculation method, full explicity is given if the information of interest is directly coded in the model.

The calculation of the *stability* of the timing model will depend on the specific type of timing behavior one is interested in. For example, if one is only interested in a specific timing invariant like the worst-case execution time, the variability of the task's execution time does not play a significant role. However, if one is interested in the ability to predict temporal trends for dynamic system optimization, like, for example, the current performance bottlenecks in a system, the *variability* among the possible execution times plays a more serious role.

By instantiating the generic concept of predictability given in Figure 2 to the time domain, we get the following definition of time-predictability:

> *Time Predictability:* The *time predictability* of a system model refers to the ability of calculating the duration of actions on the concrete system. This ability is defined in the sense of tractability rather than decidability. Thus, to have a high time-predictability, the precise calculation of the system's timing behavior must not only be decidable, but also efficient calculation techniques must be available. As a consequence, the time predictability of a model can improve as soon as more efficient calculation techniques become available.

Predictability of WCET and BCET. The above definition of time-predictability gives an intuition of what is important to make a system time-predictable. However, when it comes to the point of evaluating the predictability of specific timing properties of the system, a more exact instantiation of time-predictability is needed.

For example, in the domain of safety-critical systems, typically the WCET of a task is of primary interest to prove the timeliness of the system. As discussed above, for deciding the WCET-predictability of a system, the stability of the timing model is of minor interest. As shown in Figure 3, only the pessimism of the calculated upper bound (UB) is of interest. Thus the analyzability of the system depends on the tightness of the WCET estimate. The stability of the timing model, which could be defined as the execution-time variability as shown in Figure 3, is not directly relevant for the WCET-predictability.

What has been said about the WCET-predictability holds in an analogous form for the BCET-predictability.

3.4 Related Work on Concretizing Time-Predictability

The aim of working out a suitable understanding of time-predictability is a quite recent activity. Though time-predictability has always been the idea behind the design of real-time systems, the term time-predictability was typically used without explicit introduction. Often, architectural patterns have been introduced as time-predictable mechanisms. For example, the so-called *temporal firewall* introduced by Kopetz is a push/push interface that decouples the timing behavior of the sender and the receiver of a message [9].

Within the work of Lee et al. on the PRET processor, with the fundamental idea of giving the instruction set of a processor a time semantics, they use time-predictability in close combination with repeatability of timing [10,11].

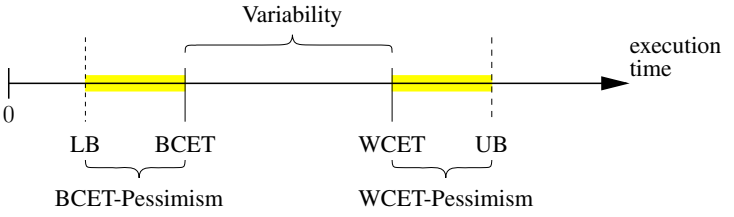

Fig. 3. The Consequences of Limited Time-Predictability

Sometimes, time-predictability was indirectly described by listing the open problems of WCET analysis and their origins. Kirner et al. have described the limitations of decomposition of WCET analysis [2]. Wilhelm et al. have identified the input-dependency and concurrence of control flow as well as the propagation of local hardware states as the major reasons for unpredictability of timing [12].

Thiele and Wilhelm have presented one of the first approaches of defining the meaning of time-predictability [4]. Translated in our terminology given in Figure 3, they described time-predictability as the pessimism of the WCET or BCET analysis. What they called "best case predictability" is the BCET-Pessimism in Figure 3, and what they called "worst case predictability" is the WCET-Pessimism in Figure 3.

A different definition of time-predictability was given by Grund [5], which was actually also the first attempt to formalize the meaning of time-predictability. The basic approach of Grund was to define the time-predictability of a program p by the relation $\frac{BCET_p}{WCET_p}$. He expressed this relation based on the set of possible system states Q and on the set of possible input values I. By denoting the execution time of a program p as $T_p(q,i)$ with $q \in Q$, $i \in I$ the time-predictability $Pr_p(I,Q)$ was defined as

$$\mathrm{Pr}_p(I,Q) \;=\; \min_{q_1,q_2 \in Q} \; \min_{i_1,i_2 \in I} \; \frac{T_p(q_1,i_1)}{T_p(q_2,i_2)} \;=\; \frac{BCET_p}{WCET_p}$$

If we translate this definition of time-predictability into our terminology given in Figure 3, it is the relative value of the execution-time variability of the program. Grund further decomposed the relative execution-time variability to the effects induced by the input data:

$$\text{State-Pr}_p(I,Q) \;=\; \min_{q_1,q_2 \in Q} \; \min_{i \in I} \; \frac{T_p(q_1,i)}{T_p(q_2,i)}$$

and to the effects induced by the hardware state:

$$\text{Input-Pr}_p(I,Q) \;=\; \min_{q \in Q} \; \min_{i_1,i_2 \in I} \; \frac{T_p(q,i_1)}{T_p(q,i_2)}$$

Compared with our definition of time-predictability, the definitions given by Thiele and Wilhelm, as well as Grund, there is a common pattern. They both represent important aspects of time-predictability, and in fact, they can be identified as some of the aspects of our definition of time-predictability. We consider the mentioned pessimism of the analysis by the *analyzability*, and the mentioned execution-time variability by the *stability* of the timing model.

3.5 On the Formalization of Time-Predictability

While the above definitions of time-predictability given by Thiele et. al and by Grund each focus on different aspects, we present a holistic definition of time-predictability that incorporates them both. With tm being the timing model that includes the hardware platform and the program to be analyzed, we get a formula for the time-predictability as follows:

$$\begin{aligned} \Pr(tm) \;=\; & analyzability(tm)^{k_a} \cdot \\ & stability(tm)^{k_s} \end{aligned}$$

To get a value for the time-predictability within the interval 0 (completely unpredictable) to 1 (maximal predictable), we would like to define the individual functions to be within the interval $0 \ldots 1$, i.e.,

$$0 \leq analyzability(tm), stability(tm) \leq 1$$

The weighting exponents k_a and k_s have to be within the range $0 \ldots \infty$. A weighting exponent of value of 1 results in a neutral weighting of the predictability ingredient, while a value smaller than 1 gives it less emphasis, and a value greater than 1 gives it more emphasis.

The stability could be defined as:

$$stability(tm) \;=\; \frac{BCET_{tm}}{WCET_{tm}}$$

The analyzability expresses the needed analysis effort as well as the achieved accuracy of the result. However, to judge the accuracy we need to define the reference result. Thus, the analyzability might only be defined for a specific instance of time-predictability. Let us in the following look at the specific instances of predictability for the WCET and BCET.

Formalization of WCET-Predictability and BCET-Predictability. The predictability of the WCET does not need to consider the stability of the timing, as it focuses only on the single WCET value. With the analog argument we can say that the predictability of the BCET does not need to consider the stability of the timing. Thus, we set the weighting exponent for the stability to zero, which indicates non-significance for the stability: $k_s = 0$. Assuming that we use the same timing model for calculating both the WCET and the BCET, we get the following formulas for WCET-predictability and BCET-predictability:

$$WCET: \; \Pr_{WCET}(tm) \;=\; analyzability_{WCET}(tm)^{k_a}$$

$$BCET: \; \Pr_{BCET}(tm) \;=\; analyzability_{BCET}(tm)^{k_a}$$

The analyzability depends on the analysis effort as well as on the accuracy of the result. As it is not obvious how to define the needed analysis effort in a way that is independent of the concrete analysis method, we ignore the analysis effort for the first and focus

only on the accuracy of the result. For the WCET-predictability it is natural to define the accuracy based on the obtained upper bound UB_{tm} of the execution time, while for the BCET-predictability we would define it based on the obtained lower bound LB_{tm}:

$$WCET:\ analyzability_{WCET}(tm)\ =\ \frac{WCET_{tm}}{UB_{tm}}$$

$$BCET:\ analyzability_{BCET}(tm)\ =\ \frac{LB_{tm}}{BCET_{tm}}$$

Summary on the Formalization of Time-Predictability. To formalize time-predictability we had to ignore the analysis effort needed to calculate the time prediction. This omission was necessary since the analysis effort can be hardly normalized, especially for different platforms and analysis methods. Furthermore, the formulas include weighting exponents to adjust the priority between analyzability and stability. For a comparison based on time-predictability it would be necessary to agree on some specific weighting exponents. Furthermore, a precise calculation of time-predictability may fail in practice as it requires to calculate precise values for the WCET and the BCET.

4 Time-Predictable Computer Systems

Given the definition of time-predictability and its instances like WCET-predictability, it is the obvious question of how systems should be built to make them time-predictable. In this section we introduce the concept of a *timing barrier*, a mechanism whose frequent use in the system design is an indicator for a good time-predictability.

First, we need to recite the concept of the *timing relevant dynamic computer state* (TRDCS), which is that part of the system's state space that has a influence on the timing behavior and that will not remain constant during all program executions [13]. Based on this, a *timing barrier* is defined as follows:

> *Timing Barrier:* The *timing barrier* is a partitioning in the time-domain of control-flow by a mechanism into two parts, called predecessor and successor, such that this mechanism reduces the fraction of the system state belonging to the predecessor that has an influence of the TRDCS belonging to the successor. Thus, a timing barrier decouples the execution time of an instruction sequence from its execution history.

This concept of a *timing barrier* was actually motivated by the concept of the *temporal firewall* as introduced by Kopetz and Nossal [9]. However, there is one fundamental difference between these concepts: while the temporal firewall decouples the execution time as well as the execution start between two programs, the timing barrier focuses only on decoupling the execution time, but not necessarily the start of an execution. In fact, the timing barrier is more a light-weight concept that is meant to be applied at many different instances within the system.

Timing barriers might be realized in software or hardware. It is important to note that a timing barrier in its generic definition does not make the execution time of the successor completely independent of the predecessor, but reduces the actual coupling. This is actually the fundamental principle of how to achieve nearly composable systems as described in Section 2. Timing barriers also tend to be useful to guide experts when selecting the hardware platform in the early design stage of a system. Even if no timing analysis exists so far, the presence of timing barriers provides hints about the level of time-predictability of the processor.

An example for a timing barrier is the flush instruction of a cache, which brings the cache into a well-defined state that is independent of the execution history. However, the timing of the successor may still depend, for example, on the pipeline state leaving the predecessor.

There are many more examples of timing barriers, and it would make great sense to collect and categorize them in order to get a better understanding of how to build time-predictable systems. Thus, we will collect concrete instances of timing barriers in our future work. By doing so we hope to get a better understanding about how to construct systems with a good time-predictability. It would also be interesting to investigate to what extent the availability of timing barriers can help to judge the time-predictability of hardware platforms.

5 Summary and Conclusion

Real-time systems have to be time-predictable to ensure their timeliness. Despite this importance of time-predictability, there is still no common agreement of what time-predictability of tasks exactly means. In this paper we therefore reviewed different definitions of time-predictability proposed so far. For our view of time-predictability we have focused on how good a timing model can get. We distinguish between the constructability of such a timing model and the time-predictability of the timing model with respect to the real system behavior. We proposed a new definition of time-predictability of code that combines the essence of previous definitions, namely the analyzability and the stability of the timing model. Further, we have also presented a formalization of this definition.

As a first proposal of how to inspect real hardware for time-predictability, we introduced the concept of a *timing barrier*. Timing barriers describe mechanisms of architectures that limit the propagation of timing effects. Using hardware patterns that implement timing barriers will help to reduce the complexity of timing analysis, which is a prerequisite for constructing systems whose timing is easily understandable and predictable.

Acknowledgments

We would like to thank the numerous colleagues who participated in the discussions about time-predictability and provided valuable feedback to our work, like, for example, at the RePP'2009 workshop in Grenoble.

References

1. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckman, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenstrom, P.: The worst-case execution time problem - overview of methods and survey of tools. ACM Transactions on Embedded Computing Systems (TECS) 7(3) (April 2008)
2. Kirner, R., Puschner, P.: Obstacles in worst-cases execution time analysis. In: Proc. 11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, Orlando, Florida, pp. 333–339 (May 2008)
3. Schoeberl, M.: Time-predictable computer architecture. EURASIP Journal on Embedded Systems (2009)
4. Thiele, L., Wilhelm, R.: Design for Timing Predictability. Real-Time Systems 28, 157–177 (2004)
5. Grund, D.: Towards a formal definition of timing predictability. In: Workshop on Reconciling Performance with Predictability, Grenoble, France (October 2009)
6. Simon, H.A.: The Architecture of Complexity. In: The Science of the Artificial, 1st edn., pp. 192–229. MIT Press, Cambridge (1969) ISBN: 0-262-69023-3
7. Kopetz, H.: Real-Time Systems - Design Principles for Distributed Embedded Applications. Kluwer, Dordrecht (1997) ISBN: 0-7923-9894-7
8. Wögerer, W.: The loss of explicity for static WCET analysis during compilation. Master's thesis, Technische Universität Wien, Vienna, Austria (2003)
9. Kopetz, H., Nossal, R.: Temporal firewalls in large distributed real-time systems. In: 6th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems (1997)
10. Edwards, S.A., Lee, E.A.: The case for the precision timed (pret) machine. In: Proc. 44th Design Automation Conference (DAC), San Diego, California (June 2007)
11. Edwards, S.A., Kim, S., Lee, E.A., Liu, I., Patel, H.D., Schoeberl, M.: A disruptive computer design idea: Architectures with repeatable timing. In: Proc. IEEE International Conference on Computer Design. IEEE, Los Alamitos (2009)
12. Wilhelm, R., Ferdinand, C., Cullmann, C., Grund, D., Reineke, J., Triquet, B.: Designing predictable multi-core architectures for avionics and automotive systems. In: Workshop on Reconciling Performance with Predictability, Grenoble, France (October 2009)
13. Kirner, R., Kadlec, A., Puschner, P.: Precise worst-case execution time analysis for processors with timing anomalies. In: Proc. 21st Euromicro Conference on Real-Time Systems, Dublin, Ireland, pp. 119–128. IEEE, Los Alamitos (July 2009)