2009 33rd Annual IEEE International Computer Software and Applications Conference

Transformational Specification of Complex Legacy Real-Time Systems via Semantic Anchoring

Yue Lu, Antonio Cicchetti, Stefan Bygde, Johan Kraft and Christer Norström
Mälardalen Real-Time Research Centre
Mälardalen University, Västerås, Sweden
{yue.lu, antonio.cicchetti, stefan.bygde, johan.kraft, christer.norstrom}@mdh.se

Abstract

RTSSim is a framework for simulating models extracted from complex legacy real-time systems which are task-oriented, run on a single processor and are developed in C. Such RTSSim models describe functional and temporal behavior as well as the resource usage of the system. However, the semantics specification of RTSSim models remains a challenging problem indeed, especially with tractable complexity to obtain a formal model which can be analyzed for instance by a model checking tool. In this paper, we present an approach towards using semantic anchoring for the transformational specification of RTSSim models, by relying on units with well-defined operational semantics and tool support. Specifically, timed automata with tasks (TAT) in TIMES is chosen as the semantic unit with the purpose of anchoring different behavioral concerns of RTSSim models in all aspects. In this respect, model transformations are conducted at the meta-model level allowing the original operational semantics of RTSSim models to be preserved, while at the same time it can be presented in TIMES models in terms of a network of TAT.

1 Introduction

Most existing embedded real-time software systems today have been developed in a traditional code-oriented manner. Many of them are maintained over extended periods of time, sometimes spanning decades, during which they become larger and more complex as a result of the iterative changes made as part of system maintenance activities. The increasing complexity makes these systems increasingly hard and expensive to maintain and verify, hence we refer to them as complex legacy systems. In our target domain, they often consist of millions of lines of C code, are task-oriented, and run on a single processor with real-time constraints.

A key challenge in model-based analysis of such large systems is how to obtain the necessary analysis model in a reasonable effort. Manual modeling would be far too time-consuming and error prone, for instance. Two methods for automated model extraction are proposed by our parallel research in [1]. A tool for automated model extraction based on program slicing [10] is in development, named *MXTC* – Model eXtraction Tool for C. The outcome of MXTC is a set of models which describe the detailed behavior on code level with respect to resource usage and interaction including for example inter-process communication (IPC), CPU time and usage of logical resources. Moreover, those models can be simulated directly in our *RTSSim* framework. The semantics specification of RTSSim models remains a challenging problem due to the use of plain C code in RTSSim modeling language. Especially, it is difficult to obtain a formal model which for instance can be analyzed by a model checking tool with tractable complexity.

Nevertheless, this issue can be attacked through a *semantic anchoring* based approach [2], i.e. by transforming RTSSim models to other models, which consist of *semantic units* (e.g. finite state machines, timed automata etc) with well-defined operational semantics and tool support. In other words, the elements and their relationships in RTSSim modeling language can be *translated* toward their counterparts at the meta-model level, i.e. the executable semantic units with well-defined behaviour, in a specific target language. The resulting model consisting of a set of semantic units, then can be subsequently used to specify and formally analyze the behavior of RTSSim models by exploiting the features of the target domain, while the original system operational semantics are preserved [5, 9]. In this paper, we present the concrete process of developing such a transformational approach to specify the behavior of RTSSim models via semantic anchoring, by choosing *timed automata with tasks (TAT)* in TIMES [4] as target language.

Paper outline The remaining part of the paper is organized as follows: in Section 2 and Section 3, we introduce some background information on RTSSim and TAT in TIMES, respectively. Section 4 describes in details how the semantic gaps are bridged via an extensive, concrete model transformation process. Finally, Section 5 concludes the paper and discusses future work.

2 System modeling of complex legacy systems in RTSSim

RTSSim is a framework with the purpose of modeling and verifying by simulation complex real-time legacy systems which are task-oriented, run on a single processor and are developed in C. RTSSim can be considered a imperative and Turing-complete domain-specific language (DSL) describing both architecture and behavior of the target system. Its syntax and semantics are as expressive as C programming language, but are extended with RTSSim primitives like message passing and task synchronization via binary semaphore, for instance.

RTSSim employs a hierarchical model to specify the system structure; going deeper, an RTSSim simulation model consists of a number of *tasks* that interact transparently with each other by IPC via message passing and task synchronization via semaphore, and the resources (e.g. message queues) there are in the system. A task may not be released for execution until a certain time (the *offset*) has elapsed after the arrival of the activating event. Each task also has a *period*, a *maximum jitter*, and a *priority*. Periods and priorities can be changed at any time by any task in the application, which is a particular feature of the system in our target domain and makes it impossible to apply the separation-of-concerns approach to address system behavior [6]. Jitter is assumed to be a variation in task release time. Offsets and arrival jitter are nonnegative and can both be larger than the period. The scheduling policy of tasks in RTSSim is fixed-priority preemptive scheduling (FPPS) and the lower numbered priorities are more significant. A task in RTSSim is composed of a number of *jobs* and invoked modeling primitives. Furthermore, the priority of the jobs inherits from the adhering task. Behavioral modeling is encapsulated in RTSSim architecture model, allowing tasks to be described by different points of view:

a) Functional behavior is brought in by means of control flow by operating on C code in RTSSim. Any valid data structures in C can be used in RTSSim, such as *pointers*. However, for simplicity, only plain integer variables, arrays, structs are applied currently;

b) Timing behavior is specified in terms of Worst-Case Execution Time (WCET) estimate on jobs and modeling primitives, which are usually over-estimated[1]. In the latter case WCET estimates are only given by static WCET analysis. On the contrary, for jobs there are two alternatives according to different applications: they can be modeled either in terms of static WCET estimates that are used in applications with hard real-time constraints, or through dynamic WCET estimates based on measurements with probability distribution when dealing with applications with soft real-time constraints. Detailed information regarding static and dynamic WCET analysis in general can be found in [11]. The abstraction of the WCET estimate is passed as argument to the "execute" primitive, which advances the global discrete-time simulation clock in RTSSim by the amount of simulation time specified. In this paper, we only consider using the static WCET estimates on jobs with the purpose of having *safe* models which capture all the timing behavior including worst-case extracted from the real system. Moreover, the preemption may only occur in the RTSSim primitives, such as "execute". They are modeled by message passing (to be introduced in the latter paragraph), while they inherit the priority from the adhering task;

c) In RTSSim, the primary intertask communication mechanism within a single CPU is *message queues*. They are shared between tasks and considered as a kind of resource usage in the system. They allow a variable number of messages, each of variable length, to be queued. A task sends a message to a message queue with the modeling primitive *sendMsg*. If the message queue is full, the sender task is blocked until there is a room available in the queue. Otherwise, the message is added to the queue's buffer. A task receives a message from a message queue with another modeling primitive, i.e. *recvMsg*. The first message is immediately dequeued and returned to the calling task in the First-In-First-Out (FIFO) pattern either with data or -1 (if the message queue is empty). Moreover, task synchronization is in terms of binary semaphore. For simplicity, task synchronization is not considered in this paper. Figure 1 shows a RTSSim task example i.e. *CTRL_TASK*.

3 Timed automata with tasks in TIMES

TAT, an extended version of timed automata with asynchronous processes (real-time tasks), i.e. computation tasks triggered by either periodically (*periodic* behavior) or by external event streams modeled through appropriate timed automata (*controlled* behavior). The main idea is to associate each location of a timed automata with an abstraction of an executable program (e.g. in C) called a *task* (or *task type*) with task parameters: fixed scheduling priority, worst-case computation time and deadline.

[1]These WCET estimates are far more than the operations e.g. addition, subtraction, multiplication, division and multiple of them would cost. It is therefore safe to remove the time consumed by the operations in order to decrease the complexity of the model.

```
void CTRL_TASK(TCB* tcb)
{
  int msg;
  int ioevent;
  int i;

  sendMessage(tcb, DCQ, MSG_GETSTS, FOREVER);

  i = 0;
  do{

    ioevent = recvMessage(tcb, IOQ, 0);

    if (ioevent > -1)
    {
      i++;
      execute(tcb, cCTRLioevent);
    }

  }while (ioevent > -1);
}
```

Figure 1. A RTSSim task example *CTRL_TASK*

As in timed automata, assume a finite alphabet Act ranged over by a, b etc. and a finite set of real-valued clocks C ranged over by x_1, x_2 etc. Let $B(C)$ ranged over by g denote the set of conjunctive formulas of atomic constraints in the form: $x_i \sim C$ or $x_i - x_j \sim D$, where x_i, x_j are clocks, $\sim \in \{\leq, <, \geq, >\}$ and C, D are natural numbers. The elements of $B(C)$ are called *clock constraints*. Syntactically, a TAT over actions Act, clocks C, and task types P is a tuple $\langle N, l_0, E, I, M \rangle$, where: N is a finite set of locations ranged over by l, m, n, l_o is the initial location, $E \subseteq N \times B(C) \times Act \times 2^C \times N$ is the set of edges, $I : N \mapsto B(C)$ is a function assigning each location with a clock constraint (a location invariant), $M : N \hookrightarrow P$ is a partial function assigning locations with task types (e.g. some of the locations may have no tasks).

Given a scheduling strategy **Sch** and a function representing the available computing resource **Run**, the semantics of an automaton $A = \langle N, l_0, E, I, M \rangle$ with initial state (l_0, u_0, q_0) is a labeled transition system with an initial state. Discrete and delay transitions are defined by the following a) and b) rules, respectively:

a) $(l,\ u,\ q) \xrightarrow{a}_{\textbf{Sch}} (m,\ u[r \mapsto 0], \textbf{Sch}(M(m) :: q))$
$\quad if\ l \xrightarrow{gar} m\ and\ u \models g$

b) $(l,\ u,\ q) \xrightarrow{t}_{\textbf{Sch}} (l,\ u + t,\ \textbf{Run}(q,\ t))$
$\quad if\ (u + t) \mapsto I(l)$

where $u \models g$ denotes that the clock assignment u satisfies the constraint g, $u + t$ denotes the clock assignment which maps each clock x to the value $u(x) + t$, $u[r \mapsto 0]$ for $r \subseteq C$ denotes the clock assignment which maps each clock in r to 0 and agrees with u for the other clocks (i.e. $C \backslash r$), $M(m) :: q$ denotes the queue with $M(m)$ inserted in q.

Withal, the standard timed automata is the underlying timed automata of TAT, of which flavor is extended with for instance finite domain integers, constant and (multidimensional) arrays of bounded integers, as well as (urgent) channels and clocks. A network of TAT is a finite set of *automata processes* composed in parallel with a CCS-like parallel composition operator [7]. The synchronization between two TAT is via a paired channels on edges, e.g. a? and a!. The functional behavior of real-time task in TIMES is specified in *task interface* in terms of for instance *Conditional (Ternary) Operator (?:)* that can be used as a shortcut for *if/else* statement. All such operations will be taken at the end of task execution. The Figure 2 and Figure 3 show two examples of TAT which are wrt. anchoring the semantic differences in RTSSim task *CTRL_TASK* referred in Figure 1. Readers can refer to TIMES tool online help for a more thorough description of TAT used in TIMES tool.

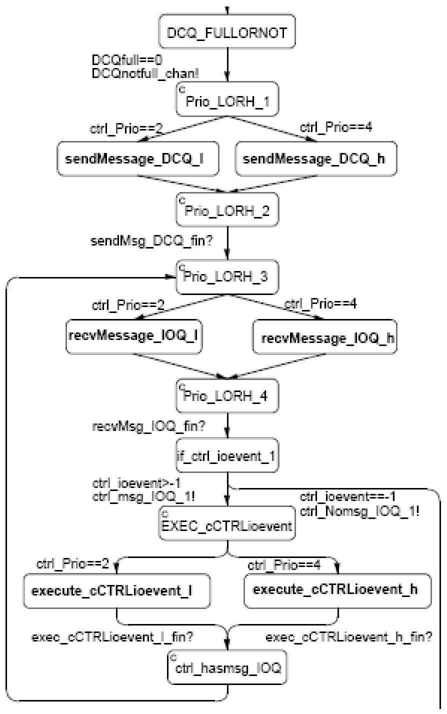

Figure 2. mcf_tat for RTSSim CTRL_TASK in Figure 1, after semantic anchoring

4 Model transformation from RTSSim to TIMES

Based on the description and discussion made in the previous two sections, the *processes* and *timed automata extended with tasks* (TAT) in TIMES can be chosen specifically as the basic architectural elements, to which RTSSim *tasks* and (subset of) C code (refers to the description in Section 2) that describes a task's behavior, can be anchored.

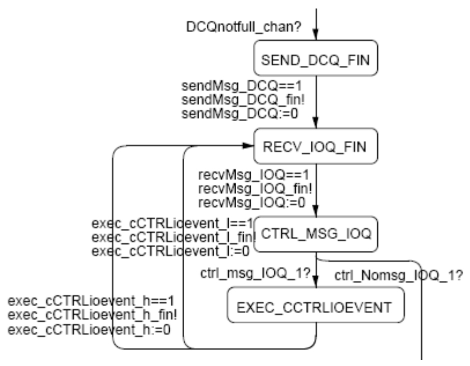

Figure 3. syn_ta for RTSSim CTRL_TASK in Figure 1, after semantic anchoring

However, since the RTSSim semantics differ from their counterparts, i.e. TAT in TIMES in all aspects listed in Table 4, hence an extensive model transformation at the meta-model level has to be performed in the interest of bridging the semantic gaps between the two models.

Due to that the priority and period of the concerning task in RTSSim can be changed at runtime by any other tasks, the model transformation between RTSSim and TIMES at the meta-model level cannot be done in a separation of a behavioral concerns approach (i.e. the model transformation wrt. system timing and functional behavior can be fully separated), which increases the complexity of the work.

The key factor to overcome the semantic differences between RTSSim models and TAT in TIMES is to identify how to model RTSSim tasks, and develop a discrete-time scheduler in TIMES which manages the execution concurrency of jobs and modeling primitives in tasks in the right preemption pattern. In order to achieve this objective, there are three modeling procedures have to be accomplished successively:

a) Finding out a way to model RTSSim tasks: representing the timing information of tasks via the abstraction of WCET estimates on jobs and intertask communication modeling primitives; modeling the special task behavior that the priority and period of the concerning task can be changed at runtime by another task; transforming task control flow which is specified in C;

b) Modeling RTSSim task parameters, i.e. encoding RTSSim task period, offset and jitter as the parameters in *automaton template* (an uninstantiated TAT and possibly some parameters) in TIMES;

c) Modeling the discrete-time scheduler in TIMES based on the previous 2 steps.

Our solutions proposed in this paper will be presented step-by-step in the following subsections.

4.1 Modeling RTSSim tasks in TIMES

4.1.1 Representing RTSSim discrete timing unit in TIMES

In RTSSim models, the time (i.e. a global simulation clock) can only be advanced by the modeling primitive "execute"[2]. While, TIMES systems evolve with continuous model-time, i.e. the model-time indeed proceeds with a constant pace. Withal, the conceptual *ticks* of clocks in TIMES can only capture the timing points at integer instants over continuum in the real-value domain. We therefore propose to map one time unit in RTSSim to one (conceptual) clock tick in TIMES, such that the temporal behavior of RTSSim jobs and modeling primitives represented as multiples of the basic timing unit can be precisely specified in TIMES as multiple of the clock ticks.

4.1.2 Modeling RTSSim task control flow in TIMES

The RTSSim tasks can be released periodically with jitter and offset, which makes to find the precise value of the hyper-period (least common multiple of tasks periods) difficult. Moreover, the times of preemption will occur at runtime when a job or modeling primitive is executing, is difficult to know. We therefore propose to model each control flow of a RTSSim task in TIMES by one TAT and one timed automata, which are referred to as *mcf_tat* (main control flow TAT) and *syn_ta* (synchronization timed automata) respectively in the following context. In each mcf_tat, the control flow of RTSSim task is specified in terms of locations which are connected with each other by *edges* (or *transitions*) between locations. Some of the locations are associated with one real-time task for each, which models either a job or a modeling primitive invoked by the RTSSim task control structure. Besides, each mcf_tat has a real-valued, local *clock* which is used to model the period of RTSSim task. The synchronization between mcf_tat and syn_ta is conducted by a set of pairs of urgent channels, i.e. $a?$ and $a!$, such that after the current running real-time tasks finishes the execution, syn_ta can drive mcf_tat to move to the next location without time elapsed. Figure 2 and Figure 3 illustrate the way of modeling the control flow of the RTSSim task *CTRL_TASK* in Figure 1 by a pair of mcf_tat and syn_ta in TIMES. In these pictures, the real-time tasks in TIMES are noted in **bold** style and the type of all the channels are *urgent*.

[2]Primitive "execute" is either invoked by RTSSim tasks, or *idle* task which is with the least significant priority and will not stop advancing the global simulation clock until there is a task to run.

Behavioral Concerns	RTSSim	TIMES
Time	The global simulation clock can be only advanced by RTSSim modeling primitive “execute” over discrete time	Continuous time in the modeling environment
Priority and period of the task on focus	Priority and period of the task on focus can be changed at runtime, and the lower numbered priority is more significant	Priority and period of real-time tasks in TAT cannot be changed at runtime, and the higher numbered priority is more significant
Communication	IPC via message passing between tasks with execution concurrency	Synchronization of TAT transitions via channels

Table 1. Behavioral gaps between RTSSim and TIMES

4.1.3 Modeling RTSSim jobs in TIMES

All the jobs in RTSSim tasks which are not concerning (i.e. the tasks of which priorities cannot be changed at runtime) are modeled in a similar way in TIMES, by one real-time task as the abstraction of WCET estimate and one auxiliary flag (i.e. bounded integer) to denote when the real-time task finishes its execution with or without being preempted. The relevant parameters of such real-time tasks in TIMES are set wrt. the following pattern: i.e. B (behavioral type of the task) is set to be *C* (controlled); C (execution time) equals to the value specified in RTSSim job; D (the relative deadline) of each job is assigned to be infinite; Pr (priority of the task) is the same value as the priority of the adhering task but in a reverse order. Moreover, each real-time task is labeled with the same name of job.

Regarding the concerning task in our target domain, its priority can only be changed from one priority to another at runtime. Except for assigning different priorities, each job in such task is modeled as a pair of two real-time tasks with the same parameter set as described before. With the intention to make such two real-time tasks more easily be identified in TIMES model, their names are consequently decorated by different postfix, i.e. either with “_l” (noted for the less significant priority) or with “_h” (noted for the more significant priority). The real-time tasks *execute_cCTRLioevent* with postfix "_l" and "_h" in the TIMES model depicted in Figure 2 can be referred to as an example.

4.1.4 Modeling RTSSim primitives on message passing

In TIMES, there is no data exchanged between processes in the adopted hand-shaking interaction mechanism, which is different from the RTSSim intertask communication behavior introduced in Section 2. A natural way to bridge the gap between the above semantic differences is to model the RTSSim communication mechanism via message passing in TIMES through shared variables and data structures. Information between TAT processes is exchanged by updating and reading from these global resources. The modeling pattern is listed as follows:

a) Each message queue in RTSSim is modeled as a bounded integer array with a bounded integer array index. The length of the array is the same as the length of the message queue in RTSSim models;

b) The RTSSim message passing modeling primitives i.e. *sendMsg* and *recvMsg* which are not invoked by the concerning task, are modeled as two real-time tasks in TIMES with the same parameters wrt. *controlled* behavioral type, constant execution time (i.e. 2 time units costed in RTSSim are anchored to 2 clock ticks in TIMES), infinite deadline, and the same priority as the adhering RTSSim task. Moreover, the described modeling pattern is used for one specific message queue. So if there is another message queue declared in the RTSSim models, accordingly another pair of two real-time tasks in TIMES which operates on that message queue has to be modeled. The only difference between each paired real-time tasks is the detailed functional behavior wrt. operating the certain message queue in the task interface. If such modeling primitives are used in the concerning task, then each of them will be modeled as two real-time tasks in the similar way as modeling jobs in the concerning task, i.e. with the same parameters and functional behavior but different priorities;

c) The operations specified in the task interface on adding or retrieving a data from a FIFO message queue can be easily achieved with the help from *Conditional (Ternary) Operator (?:)*. Due to limited space, it is not described in details.

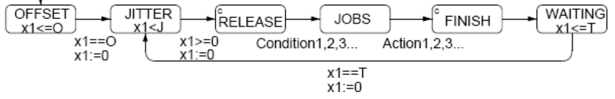

Figure 4. Encoding RTSSim task parameters as the parameters in automaton template in TIMES

4.2 Modeling RTSSim task parameters in TIMES

Except for the task priority issue which has been resolved in Subsection 4.1.3, the other RTSSim task parameters i.e. period, jitter and offset are encoded in automaton templates in TIMES as three parameters *T* (period), *O* (offset) and *J* (jitter) with the type *const* respectively. Moreover, the period of the concerning task which can be changed at runtime by other tasks, is modeled as a global bounded integer with the interval [min_period, max_period] coming from the requirements. Consequently, the value of such global bounded integers can be easily changed by for instance an action on an edge in mcf_tat which models another RTSSim task. Refer to Figure 4 as an example.

4.3 Modeling the discrete-time scheduler in TIMES

According to the modeling pattern described previously regarding RTSSim tasks and its parameters, the effort spent on modeling the RTSSim discrete-time scheduler can be largely eased by combiningly using an encoding FPPS scheduler (using two clocks) in TIMES presented in [3], where task queue and operations on the queue related to the given scheduling strategy are encoded as a timed automaton (called the scheduler). Once given the parameters and arrival pattern of real-time tasks under the specific scheduling strategy, the tasks execution concurrency can be controlled automatically in TIMES, e.g. when to release, be preempted, resume and finish the execution. This is also one motivate for using TIMES in this work, rather than UPPAAL [8], with the scope of analyzing the complex legacy system which is task-oriented and running on a single processor. Moreover, the scheduling policy adopted in this work is *user-defined priorities preemptive scheduling* in TIMES.

5 Conclusions and future work

In this paper we present a transformational approach towards semantics specification of complex legacy real-time systems which are task-oriented, run on a single processor and are developed in C. I.e. via semantic anchoring, the RTSSim models (subset of C code) extracted from complex legacy systems based on program slicing, are transformed to a network of timed automata with tasks (TAT) in TIMES with original semantics preserved. One important continuation work is to apply the formal analyzable model in TIMES obtained in this work to the research wrt. WCET analysis of tasks with intricate dependencies through shared variables. An industrial evaluation of RTSSim model extraction is in progress together with ABB Robotics and in future work we intend to investigate the proposed approach on RTSSim models generated from that system. Furthermore, anchoring other semantic differences such as task synchronization and more complex data structures in C, will be also considered.

Acknowledgment

This work was supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS. Withal, we are grateful to Dr. Cristina Seceleanu for comments and improvement suggestions.

References

[1] J. Andersson, J. Huselius, C. Norström, and A. Wall. Extracting simulation models from complex embedded real-time systems. In *Procs. of the Int. Conf. ICSEA'06*. IEEE, 2006.

[2] K. Chen, J. Sztipanovits, S. Abdelwahed, and E. Jackson. Semantic anchoring with model transformations. In *Procs. of ECMDA-FA 2005*, Nuremberg, Germany, November 2005.

[3] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theor. Comput. Sci.*, 354(2):301–317, 2006.

[4] E. Fersman, P. Pettersson, and W. Yi. Timed automata with asynchronous processes: schedulability and decidability. In *Procs. of TACAS 2002*, pages 67–82. Springer-Verlag, 2002.

[5] Xu Ke, P. Pettersson, K. Sierszecki, and C. Angelov. Verification of comdes-ii systems using uppaal with model transformation. In *Procs. of the Int. Conf. RTCSA08*. IEEE Computer Society Press, 2008.

[6] Xu Ke, K. Sierszecki, and C. Angelov. Comdes-ii: A component-based framework for generative development of distributed real-time control systems. In *Procs. of the Int. Conf. RTCSA '07*, pages 199–208. IEEE Computer Society, 2007.

[7] R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., 1989.

[8] Uppaal, www.uppaal.com, 2008.

[9] A. Vallecillo. A Journey through the Secret Life of Models. In *Perspectives Workshop: MECS*, number 08331 in Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008.

[10] M. Weiser. Program Slicing. In *Proc. of the Int. Conf. ICSE'81*, pages 439–449. IEEE Press, 1981.

[11] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *Trans. on Embedded Computing Sys.*, 7(3):1–53, 2008.