FORMAL VERIFICATION OF OSEK/VDX BASED APPLICATIONS

Libor Waszniowski, Zdenek Hanzalek

Czech Technical University
Faculty of Electrical Engineering, Department of Control Engineering
Karlovo nám. 13, 121 35 Prague 2, Czech Republic
{xwasznio, hanzalek}@fel.cvut.cz

Abstract: This article shows, how a preemptive multitasking application running under a real-time operating system compliant with OSEK/VDX standard can be modeled by timed automata. The application under consideration consists of several tasks, it includes synchronization by events and resource sharing. For such system, model-checking theory based on timed automata and implemented in model-checking tools can be used to verify time and logical properties of the proposed model. It is shown that the proposed model is over-approximation in the case of preemptive scheduling policy.

1 INTRODUCTION

This paper deals with modeling of applications running under real-time operating system (OS). Typical application under assumption is a controller consisting of periodic and aperiodic tasks constrained by deadlines and synchronized via communication primitives.

The model-checking (Larsen, *et al.*, 1995) approach, shown in this paper, provides timed automata (Alur and Dill, 1994) model of an operating system, application tasks and controlled environment. In the scheduling theory, the task model usually consists of its execution time, the blocking time and the inter-arrival time. Our approach assumes a *fine grain model* of the task internal structure consisting of computations, system calls, selected variables, code branching and loops. Therefore the model combines both, logic and timing parameters of a discrete event system enabling to check rather complex properties (safety and bounded liveness properties, schedulability, state reachability) by model-checking tools (e.g. UPPAAL[1] (Behrmann, *et al.*, 2001) and Kronos[2] (Daws, *et al.*, 1996)) in finite time.

Even though timed automata and model-checking (analogous to other formal methods) allows modeling and verifying almost everything, it is generally known, that they are susceptible to state space explosion. This fact restricts the size of verified application to the small size that seems to be unusable in praxis (compared with matured schedulability analysis methods (Liu, 2000)). Therefore we try to show in this paper, how to build a compromise model of a reasonable size on one side and of reasonable granularity allowing detailed formal analysis of real-time properties that can not be done by schedulability analysis on the other side.

Methods for schedulability analysis, e.g. rate monotonic analysis (RMA) ((Sha, *et al.*, 1991; Liu, 2000) have been widely used in praxis. However they can lead to pessimistic results when non-periodic tasks, shared resources and other features are incorporated (Bailey, *et al.*, 1995). The schedulability analysis based on model-checking of fine grain model provides less pessimistic results in some cases (Waszniowski and Hanzálek, 2003).

Fersman, *et al.* in (2002) and (2003) extended timed automata by asynchronous tasks (i.e. tasks triggered by events) to provide model for event-driven systems. This approach provides good results for aperiodic tasks but it is not suited to model the task internal structure as follows from results of (Krčál and Yi, 2004). Corbet in (1996) provides model of real time Ada tasking programs based hybrid automata. Opposite to timed automata used in our approach, reachability problem is undecidable for hybrid automata and therefore the verification algorithm termination is not guarantied in general. Timed automata are used to model primitives of Ravenscar run-time kernel for Ada in (Lundqvist and Asplund, 2003). However, the time in application is discrete opposite to our approach where the time is dense.

This paper is organized as follows: Section 2 describes *fine grain model* used in this paper. Sections 3 and 4 present timed automata models of tasks and OSEK compliant OS (OSEK, 2003). This model is an over-approximation from the model-checking point of view in some cases as it is shown in Section 5.

2 APPLICATION FINE GRAIN MODEL

Fine grain model treats tasks and interrupt service routines (ISR) internal structure, the OS functionality and the controlled environment behavior. All components are modeled by timed automata

[1] http://www.uppaal.com
[2] http://www-verimag.imag.fr/TEMPORISE/kronos/

synchronized via channels and by shared variables. The task model consists of several blocks of code called *computations* (characterized by their BCET and WCET), calls of OS services, selected variables, and code branching and loops (affected by values of selected variables).

Structure of the entire model is on Fig. 2.1. Rectangular blocks represent particular timed automata. Synchronization is expressed by arcs labeled by name of the synchronization channel. The most important data structures are shown in the right side of the figure. The essential components are explained in the following sections.

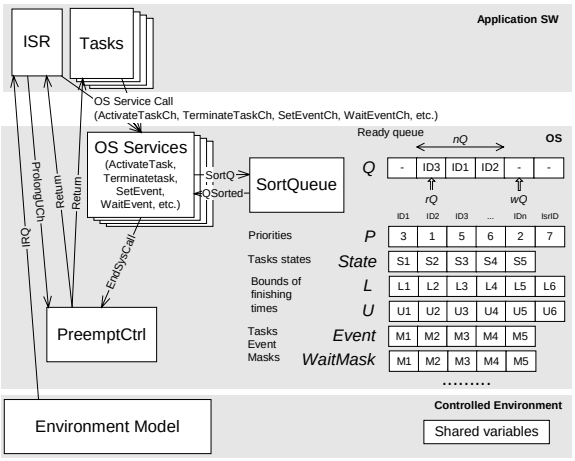


Fig. 2.1 Overview of entire timed automata model

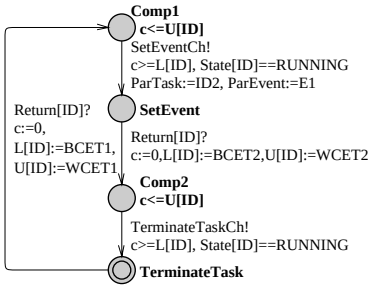
When a general property of the model is analyzed by exhaustive state space search (done by model checking tool), an execution time of a task must be specified by an interval covering all possible cases, i.e. ⟨BCET, WCET⟩. Due to scheduling anomaly, WCET of *computations* do not necessary lead to the worst case finishing time of the whole task.

3 TASK MODEL BY TIMED AUTOMATA

Each task (or its instance) is modeled by one timed automaton in UPPAAL notation (see Fig. 3.1). Double circle represents initial location. A location can be labeled by its name and a time invariant in the form "*c<=U*", allowing to stay in the location only when the clock *c* is smaller or equal to *U*. A transition can be labeled by a synchronization (channel name with '?' or '!'), a guard (comma separated logical terms and an assignment (comma separated assignments by ':=').

Fig. 3.1 shows an example of a task executing *computations Comp1* and *Comp2* and calling OS services *SetEvent(task,event)* and *TerminateTask*. Each *computation* is represented by one location of the same name (e.g. *Comp1*). The time spent in this location (measured by clock *c*) represents *computation's* finishing time (including preemption) and it is bounded by values stored in integers *L[ID]* and *U[ID]* (elements of arrays *L* and *U* respectively, where index *ID* is unique tasks identifier). These bounds are initialized to BCET and WCET, and they are increased when the task is preempted (provided by timed automaton *PreemptCtrl* described later).

```
Task1()
{
  Comp1;
  SetEvent(ID2,E1);
  Comp2;
  TerminateTask();
}
```



a) Pseudo-code b) Task automaton

Fig. 3.1 Simple task example

OS services calls are modeled by transitions synchronized by channels of corresponding names (e.g. *SetEventCh!*) and by locations of corresponding names (e.g. *SetEvent*) where the task is waiting return from services (channel *Return[ID]?*).

4 OS KERNEL MODEL

The OS model consists of variables representing OS objects, and timed automata representing OS services, managing preemption (*PreemptCtrl*) and sorting ready queue according to priorities (*SortQueue*).

Each OS service is modeled by timed automaton representing its functionality defined by OSEK specification (OSEK, 2003). The automaton is waiting in its initial state until its function is called from the task model. Then it manipulates tasks states, the ready queue (*Q*) and other operating system objects (e.g. events) and chooses the highest priority task to run and store its *ID* in variable *RunID*. Then it invokes *PreemptCtrl* automaton (by channel *EndSysCall*) modeling the context switch and providing a preemption control. As an example of a service model we introduce *WaitEvent(Mask)* service that cause the task wait for events in *Mask*. Fig. 4.1 shows *WaitEvent* OS service functionality in a pseudo-code.

```
WaitEvent (Mask)
{
  if ((Event[RunID] & Mask) == 0)
  {
    State[RunID] := WAITING;
    WaitMask[RunID] := Mask;
    Release Internal Resource;
    RunID := Extract Top of ReadyQ;
    ContextSwitch;                    // modeled in PreemptCtrl
    Get Internal Resource;
    State[RunID] := RUNNING;          // modeled in PreemptCtrl
  }
  return E_OK;
};
```

Fig. 4.1 *WaitEvent* pseudo-code

WaitEvent OS service automaton is depicted in Fig. 4.2. Locations marked by "**c**" are so called committed locations in UPPAAL notation. It must be left immediately, without any interference of other

automaton that is not in a committed location. Since all locations in the automaton in Fig. 4.2, except the initial one, are committed locations, therefore the whole service seems to be atomic from the point of view of tasks and controlled environment models.

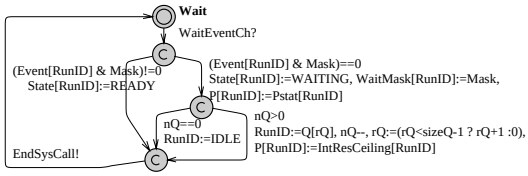


Fig. 4.2 *WaitEvent* service automaton

PreemptCtrl automaton, depicted in Fig. 4.3, starts execution of scheduled task (*RunID*) and provides prolongation of finishing time bounds (*L[ID]* and *U[ID]*) of preempted tasks. If the task that should be scheduled now (*RunID*) was preempted by a task released by an ISR in the past, its state has been *PREEMPTED* since then (set in the ISR model). In this case, the *RunID* task model is in the location corresponding to some *computation* and its progress must be allowed now by setting its state *RUNNING* in *PreemptCtrl*. If the *RunID* task model waits for synchronization *Return[RunID]* in a location corresponding to an OS service call (its state is *READY*), its state is also set *RUNNING*, and the progress in the task model is allowed by the synchronization via channel *Return[RunID]*. Since a new *computation* is started in *RunID* task in this case, bounds *L[i]* and *U[i]* of all *PREEMPTED* tasks (i.e. all tasks that are in location corresponding to a *computation*) are moreover increased by bounds of the currently beginning *computation* (*L[RunID]* and *U[RunID]*).

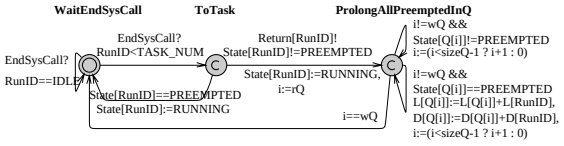


Fig. 4.3 *PreemptCtrl* automaton (interrupts are omitted)

5 MODEL OVERAPPROXIMATION

When the preemption occurs the finishing time bounds *L[Preempted]* and *U[Preempted]* of the preempted *computation* should be prolonged by the duration of the preemption. Since the right duration of the preemption cannot be measured in timed automata (a clock variable cannot be stopped or stored), the bounds *L[Preempted]* and *U[Preempted]* are increased by bounds of the possible preemption that are *L[Preempting]* and *U[Preempting]*, the finishing (execution) time bounds of the preempting task *computation*. This introduces an additional non-determinism to the model since the duration of the preempted task preemption is not necessary equal to the duration of the preempting task execution (what holds in the real system). Therefore the set of real system behaviors is subset of the modeled behaviors, i.e. the model is an over-approximation.

To illustrate the over-approximation let us consider for example low-priority task T_{low} with execution time $C_{low} \in [1,4]$ preempted by high-priority task T_{high} with execution time $C_{high} \in [2,4]$. All possible relative finishing times of both tasks in the real system and in the proposed model are depicted in Fig. 5.1. Finishing time of T_{high} is always equal to its execution time C_{high}. Finishing of T_{low} is equal to its execution time C_{low} plus preemption duration. Preemption duration is bounded by bounds of C_{high} in the model but it is equal to the actual finishing time of T_{high} in the real system.

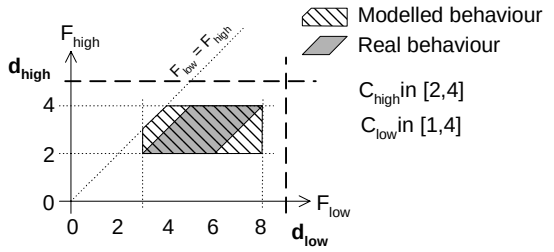


Fig. 5.1 Possible values of relative finishing times *F* of preempting task T_{high} and preempted task T_{low}

Fig. 5.1 shows that not all modeled behaviors can occur in the real system. It is very important to keep this fact in mind during the verification process, since the over-approximation does not preserve a general property. It means that it cannot be automatically concluded that a general property satisfied by the model is also satisfied by the real system. It is important from the practical point of view, that over-approximation preserves safety and bounded liveness properties (Berard, *et al.*, 2001). A safety property states that, under certain conditions, an undesirable event never occurs. A bounded liveness property states that, under certain condition, some desirable event will occur within some deadline. Fig. 5.1 also shows that the worst case finishing time of each task is the same in the model and in the real system. Result of the schedulability analysis based on this model is therefore correct and corresponds to reality (it is not pessimistic).

6 CONCLUSION

We have demonstrated, how timed automata can be used for the multitasking preemptive application modeling. Even though the model is an over-approximation of the real system, complex time and logical properties considering application data and controlled system model can be verified by model-checking tool, since safety and bounded liveness properties (the most important groups) are preserved by an over-approximation.

Opposite to hybrid automata allowing precise modeling of the preemption (Corbet, 1996), termination of the verification algorithm is guaranteed for timed automata. Opposite to models based on timed automata extended by tasks (Fersman, *et al.*, 2002), the internal structure of the preemptive task can be modeled. Opposite to models used in schedulability analysis, a timed automata based model consider the task internal structure and the controlled environment. Consequently the less pessimistic analysis is provided by model-checking, especially when the analyzed application contains features that make the response time analysis pessimistic (e.g. tasks self-suspension).

Off course, an exhaustive analysis of the detailed timed automata model subjects to a state space explosion (what is a general property of most formal methods (Corbet, 1996)). Therefore the proposed model is abstract as much as possible and contains only information necessary for a correct verification. The operating system model use only modest data structures, it does not use any clock variables (duration of OS services and context switch is involved in the execution time of *computations*), it does not allow any non-determinism and all locations are committed what prevents paths interleaving and therefore restricts explored state space. Notice also that OSEK is one of the most appropriate operating systems to be modeled by timed automata since it is static (all objects are created at the compilation time) and it is designed for a modest runtime environment of embedded devices. The model of application tasks must be designed as a compromise between the model precision and its state space size. It is necessary to limit the size of modeled data, non-determinism and number of tasks and *computations* to obtain a model of reasonable size.

Even for these restrictions, model-checking approach is applicable for formal verification of realistic applications whose verification done manually by human would be hard and error prone.

ACKNOWLEDGEMENT

This work was supported by the Ministry of Education of the Czech Republic under Project 1ET400750406.

REFERENCES

Alur, R. and D.L. Dill (1994). A theory of timed automata. *Theoretical Computer Science*, **126**, 183-235.

Bailey C.M., A. Burns, A.J. Wellings and C.H. Forsyth (1995). A Performance Analysis of a Hard Real-Time System. *Control Engineering Practice*, **3**(4), 447-464.

Behrmann, G., A. David, K.G. Larsen, O. Möller, P. Pettersson and W. Yi (2001). Uppaal - Present and Future. In: *Proceedings of the 40th IEEE Conference on Decision and Control (CDC'2001)*. pp. 2881-2886, Orlando, Florida.

Berard, B., M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen and P. McKenzie (2001). *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer Verlag.

Corbett, J. C. (1996). Timing analysis of Ada tasking programs. *IEEE Transactions on Software Engineering,* **22**(7), pp. 461-483.

Daws, C., A. Olivero, S. Tripakis and S. Yovine (1996). The tool Kronos. In: *Proceedings of Hybrid Systems III, Verification and Control*, LNCS 1066, 208-219. Springer-Verlag, New York.

Fersman, E., P. Pettersson, and W. Yi (2002). Timed Automata with Asynchronous Processes: Schedulability and Decidability. In: *Proceedings of 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2002*, LNCS 2280, pp.67-82, Springer-Verlag

Fersman, E., P. Pettersson, and W. Yi (2003). Schedulability Analysis using two clocks. In: *Proceedings of TACAS'03*, LNCS 2619 pp 224-239. Springer-Verlag.

Klein, M., T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour (1993). *A Practitioner's Handbook for Real-Time Systems Analysis*. Kluwer Academic Publishers, Boston.

Krčál, P. and W. Yi (2004): Decidable and Undecidable Problems in Schedulability Analysis Using Timed Automata. In: *Proceedings of TACAS'04*, LNCS 2988, pp 236-250. Springer-Verlag.

Larsen, K.G., P. Pettersson, and Yi, W. (1995). Model-Checking for Real-Time Systems. In *Proceedings of the 10th International Conference on Fundamentals of Computation Theory*, LNCS 965, 62-88. Springer Verlag

Liu, J.W.S. (2000). *Real-time systems*. Prentice-Hall, Inc., Upper Saddle River, New Jersey

Lundqvist, K. and L. Asplund (2003). A Ravenscar-Compliant Run-time Kernel for Safety-Critical Systems. *Real-Time Systems Journal*, **24**(1): 29-54.

OSEK (2003). *OSEK/VDX Operating System Specification 2.2.1*. http://www.osek-vdx.org/

Sha, L., M. Klein and J. Goodenough (1991). Rate Monotonic Analysis for Real-Time Systems. 129-155. Foundations of Real-Time Computing: Scheduling and Resource Management. Kluwer Academic Publishers, Boston.

Waszniowski, L. and Z. Hanzálek (2003). Analysis of Real Time Operating System Based Applications. In: *Proceedings of the 1st International Workshop on Formal Modeling and Analysis of Timed Systems, FORMATS'03*, LNCS 2791, pp. 219 - 233. Springer-Verlag Heidelberg.