Modeling and Verification of Real-Time Embedded Systems

Mustapha BOURAHLA
Computer Science Department, University of M'Sila, Algeria
mbourahla@hotmail.com

In this paper, we present an approach for the analysis of real-time embedded systems. The analysis is based on the verification of the schedulability of tasks in addition to the verification of requirements on the overall system. The real-time embedded system is interacting with an environment composed essentially from a plant and other computerized software. In our approach we identify the behaviour of the plant using observed output from a simulation or from a real system. This measured/observed output will be used to model the behaviour of the plant. The specification of tasks controlling the plant will be integrated to this model. At the end, a model representing the scheduler will be composed with the plant models to get an overall system model which will be used for analysis. We have used the formalism of timed automata extended to specify tasks and scheduling which are implemented within a tool called "TIMES" for real-time model checking.

Real-Time Embedded Systems, Modeling and Verification, Timed Automata, Scheduling.

1. INTRODUCTION

An embedded system is a special-purpose computer system designed to perform dedicated functions, often with real-time constraints that must be met for reasons such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs. An embedded system is not always standalone device; it is usually embedded as a small computerized part of a complete larger device including hardware and mechanical parts. For example, an embedded system in an automobile provides a specific function as a subsystem of the car itself. The program instructions written for embedded systems are referred to as firmware, and are stored in read-only memory or flash memory chips. They run with limited computer hardware resources: little memory, small or non-existent keyboard and/or screen. Embedded systems control many of the common devices in use today. Embedded systems span all aspects of modern life and there are many examples of their use. Like mobile phones, anti-lock braking system (ABS) and many others all of which use embedded devices for sensing and controlling. Embedded systems range from no user interface at all to complex graphical user interfaces that resemble modern computer desktop operating systems. Simple embedded devices use buttons, LEDs, and small character/digit-only displays, often with a simple menu system. Embedded systems talk with the outside world via peripherals. Embedded systems often reside in machines that are expected to run continuously for years without errors. In some cases they recover by themselves if an error occurs. Therefore the software is usually developed and tested [5, 9] more carefully than that for personal computers, and unreliable mechanical moving parts such as disk drives, switches or buttons are avoided.

There are several different types of software architecture in common use. Most embedded systems are predominantly interrupt controlled. This means that tasks performed by the system are triggered by different kinds of events. An interrupt could be generated by a serial port controller receiving a byte. This architecture is based on a multitasking kernel with discrete processes [7]. In this type of system, a low-level piece of code switches between tasks or threads based on a timer (connected to an interrupt). This is the level at which the system is generally considered to have an "operating system" kernel. Depending on how much functionality is required, it introduces more or less of the complexities of managing multiple tasks running conceptually in parallel. As any code can potentially damage the data of another task programs must be carefully designed and tested [9], and access to shared data must be controlled by some synchronization strategy, such as message queues, semaphores or a non-blocking synchronization scheme. Because of these complexities, it is common for organizations to buy a real-time operating system, allowing the application programmers to concentrate on device functionality rather than operating system services, at least for large systems; smaller systems often cannot afford the overhead associated with a generic real time system, due to limitations regarding memory size, performance, and/or battery life. A microkernel is a logical step up from a real-time operating system. The usual arrangement is

that the operating system kernel allocates memory and switches the CPU to different threads of execution. User mode processes implement major functions such as file systems, network interfaces, etc.

The development of the embedded systems is a major task. It is necessary to ensure that it works correctly in its real operating environment. Due to lack of development resources it is not feasible to validate the system for all possible environment behaviors [11, 12]. In this paper, we present an approach to model and verify embedded real-time systems using the tool “TIMES” [3, 4, 14] developed upon the tool “UPPAAL” [13] which is a mature integrated tool environment for modeling, verification, simulation, and testing of real-time systems modeled as networks of timed automata [2]. Our contribution is suitable for helping the development of embedded real-time systems. We will consider an embedded system as a system interacting closely with its environment which typically consists of the controlled physical equipment (the plant) accessible via sensors and actuators. It can also interact with other computer based systems or digital devices accessible via communication networks using dedicated protocols, and human users.

Our approach uses the formalism of timed automata extended to specify tasks and their scheduling [1]. A timed automaton [2] is an extended finite state machine equipped with a set of special real-valued variables called clocks whose value automatically increase as time progresses. Clocks may be used in enabling conditions on transitions and may be reset when a transition is executed. In addition, a clock invariant condition in a location limits how long the automaton may remain in that location. Before modeling the embedded system, the environment (the plant) behavior should be identified using collected data from observed real system or from simulation of a system specification. This plant identification will be represented as a model which will be extended with specified tasks controlling its behavior. The result model will be specified as an extended timed automaton. The embedded system is modeled by another extended timed automaton. The annotated composition of all these extended timed automata will yield a timed automaton that is ready for simulation and verification of properties and schedulability analysis using the integrated model checking algorithm.

The rest of the paper is organized as follows. In Section 2, we present our technique for modeling real-time embedded systems through an example. Section 3 presents how the developed model can be used for simulation and verification of requirements and schedulability acceptance. At the end, a conclusion is given.

2. MODELING EMBEDDED SYSTEMS

We will present our technique for modeling and then verifying embedded systems. This technique of modeling is based on extended timed automaton which is used by the tool “TIMES” developed upon the tool “UPPAAL” to facilitate the modeling and analysis of scheduling tasks. It is appropriate for systems that can be described as a set of preemptive or non-preemptive tasks which are triggered periodically or sporadically by time or external events. It provides a graphical interface for editing and simulation, and an engine for schedulability analysis and it uses the “UPPAAL” verifier for verification of real-time temporal properties.

The plant model should be identified before modeling the embedded system. A corresponding estimated model can be computed from observed real plant system or from a simulated mathematical specification. When the plant model is identified, a specification based on extended timed automata will be realized for verification of design requirements and scheduling acceptance using the technique of model checking.

Figure 1 illustrates the architecture we use for embedded systems integrates to its environment. This architecture is based on a structure composed of three parts: the hardware (physical/mechanical) part (or the plant), the micro-kernel (real time preemptive operating system) for scheduling tasks, and the (real-time) application composed of several tasks. The tasks are periodic, sporadic or controlled (a-periodic).

FIGURE 1: Architecture of embedded system

The physical part is the device to be controlled. It is usually a complex (composed) device. Many components will randomly interact (or we can say their behaviors mutually affect each of other). This complexity of behavior is difficult to model. Thus, we need to model the behavior of

each component separately and then combine (compose) the models to obtain the overall complex behavior. The following figure (Figure 2) illustrates this decomposition.

FIGURE 2: Decomposition of the plant

Each physical component will be controlled by a subset of tasks. The executions of the tasks are scheduled by a scheduler model specified with a defined strategy (HPF: Highest Priority First, Earliest Deadline First, Least Laxity First, etc...) taking as arguments the task priority, worst execution time, deadline, etc.

2.1 Identification and specification of the plant

A dynamic system is a system where its output depends on the past inputs and past outputs at many previous time instants. Models describe relationships between measured inputs u , measured output y and unmeasured inputs, called disturbance inputs or noise e . All these signals are functions of time, and the value of the input at time t will be denoted by $u(t)$. Often, in the identification context, only discrete-time points are considered, since the measurement equipment typically records the signals just at discrete-time instants, often equally spread in time with a sampling interval of T time units. The modeling problem is then to describe how the three signals relate to each other. State-space models are common representations of dynamic models. The state-space representation looks like

$$\begin{aligned} x(t+1) &= A * x(t) + B * u(t) + K * e(t) \\ y(t) &= C * x(t) + D * u(t) + e(t) \end{aligned} \quad (1)$$

Here $x(t)$ is the vector of state variables. The model order is the dimension of this vector. The matrix K determines the disturbance properties. Notice that if $K = 0$, then the noise source $e(t)$ affects only the output, and no specific model of the noise properties is built. This is usually referred to as an output-error model. Notice also that $D = 0$ means that there is no direct influence from $u(t)$ to $y(t)$. Thus the effect of the input on the output all passes via $x(t)$ and will thus be delayed at least one sample. The first value of the state variable vector $x(0)$ reflects the initial conditions for the system at the beginning of the data record. When dealing with models in state-space form, a typical option is whether to estimate D , K , and $x(0)$ or to let them be zero.

As the plant which is a dynamic system, has in general complex unknown behavior, it is necessary to build mathematical models of this dynamic system based on measured data (taken from a simulation or behavior of a real system). Essentially by adjusting parameters within a given model until its output coincides as well as possible with the measured output. A good test is to take a close look at the model output compared to the measured one on a data set that was not used for the fit ("Validation Data"). It is also valuable to look at what the model could not reproduce in the data ("the residuals"). This should not be correlated with other available information, such as the system input.

The identification process amounts to repeatedly selecting a model structure, computing the best model in the structure, and evaluating the model properties to see if they are satisfactory. The cycle can be itemized as follows: (1) Design an experiment and collect input-output data from the process to be identified. (2) Examine the data. Polish it so as to remove trends and outliers, and select useful portions of the original data. Possibly apply filtering to enhance important frequency ranges. (3) Select and define a model structure (a set of candidate system descriptions) within which a model is to be found. (4) Compute the best model in the model structure according to the input-output data and a given criterion of fit. (5) Examine the obtained model properties. (6) If the model is good enough, then stop; otherwise go back to Step 3 to try another model set. Possibly also try other estimation methods (Step 4) or work further on the input-output data (Steps 1 and 2).

2.2 Example

Our example is a water-heating system. The plant is composed of a tube containing material for heating water that is fed from a valve. The water volume should be always kept between two levels (Low and High). The water temperature should be also controlled to not be too cold or hot. Thus, this system is controlled by an embedded computer system equipped by a real-time application composed of four tasks: *fill()* to open the valve for letting water fed into the tube, *empty()* to close the valve, *heat()* to let heating system working and *cool()* to stop it. These tasks are scheduled by a real time kernel installed within this system.

2.2.1 Design and simulation of the experiment

To produce a state space model for our example, we have designed and simulated an experiment using the “Matlab” toolbox “SimuLink”. This design is based on a high level mathematical specification which describes the behavior of this system. The change of water volume can be specified as

$$\frac{dV(t)}{dt} = F(t) - k_0 * e^{\frac{-E}{R*T(t)}} \quad (2)$$

Where the water quantity $F(t)$ fed into the tube from the valve is added to the current volume $V(t)$ and then we subtract the quantity of water vaporized due to temperature, which states that the rate of vaporization increases exponentially with the absolute temperature. k_0 is here an unknown no thermal constant, E is the activation energy, R is Boltzman's ideal gas constant and $T(t)$ the temperature in the tube. Similarly, using the energy balance principle, the temperature change per time unit $\frac{dT(t)}{dt}$ in the tube can be modeled as

$$\frac{dT(t)}{dt} = \frac{F(t)}{V(t)} * (T_f(t) - T(t)) - \frac{U * A}{C_p * \rho * V(t)} * (T(t) - T_j(t)) \quad (3)$$

Where the two terms describe changes due to that the feed stream temperature $T_f(t)$ and the material coolant temperature $T_j(t)$ differ from the water temperature, respectively. In this equation, C_p is a heat capacity term, ρ a density term, U an overall heat transfer coefficient and A the area for the heat exchange (coolant/tube area). This mathematical description of the example can be designed to be simulated using an appropriate tool. We have used the simulator toolbox “SimuLink” integrated to the “MatLab” tool to design and simulate this system. The values given to different constants are as follows.

$$k_0 = 35e3, E = 11850, R = 19.858, U * A = 145, \text{ and } C_p * \rho = 480 \quad (4)$$

The simulation results are collected to be used for identifying and then modeling the plant as a state space model. The figures 3 and 4 below show these data.

FIGURE 3: Water level changes

FIGURE 4: Temperature level changes

2.2.2 Specification of simulation results

The data produced from the simulation should be examined and polished so as to remove trends and outliers. Then, we select useful portions of the original data by applying filtering to enhance important frequency ranges. Using the model structure presented in Section 2.1, the best model is computed in the model structure according to the input-output data and a given

criterion of fit. The simulation results represent the behavior of two separated components, the first is used to control the volume of water (we will call it "WaterVolume") and the second is used to control its temperature ("WaterTemperature"). It is obvious that the two components affect each other. The system identification toolbox of "Matlab" is used for identifying the two models for the components "WaterVolume" and "WaterTemperature". The state space models of the component "WaterVolume" and the component "WaterTemperature" are described as in the equation 5 and the equation 6, respectively.

$$\begin{aligned} x(0) &= 0 \\ x(t+1) &= x(t) + 0.9 * F(t) - 0.02 * T(t) \end{aligned} \quad (5)$$

$$V(t) = x(t)$$

$$\begin{aligned} y(t) &= 0 \\ y(t+1) &= y(t) + 0.07 * T_j(t) - 0.01 * V(t) \\ T(t) &= y(t) \end{aligned} \quad (6)$$

Then, we examine their properties by comparison with unused data. Thus, we have used data in the interval $[4500, 4900]$ which is not used to identify the state space models. Figure 5 shows the comparisons results. The output of the first state model representing the changes of water level is close by more than 86% to the measured output. The output of the second state model is close by more than 90%.

FIGURE 5: Comparisons of measured outputs with simulated model outputs

These two state space models representing interacting components of the plant will be used as a base for modeling the embedded system.

2.3 Modeling the example embedded system

The plant components will be represented by processes (network of processes). Each process behavior is described by a timed automaton extended with data variables and tasks. The behavior of each plant component is guided by actions labeling the transitions. The values of variables used in the different actions are controlled by the tasks behaviors. The transitions actions are defined according to the state space models of the plant components which are defined using observed or measured data as described in the section above. A location of an extended automaton may be annotated with a task or a set of tasks that will be triggered when the transition leading to the location is taken. The triggered tasks will be put in a task queue (i.e. ready queue in operating system) and scheduled to run according to a given scheduling policy. The scheduler should make sure that all the task constraints are satisfied in scheduling the tasks in the task queue. To model concurrency and synchronization between automata, networks of automata are constructed in the standard way with the annotated sets of tasks on locations.

Thus, a system description consists of a task set and a network of timed automata extended with the tasks. For our example, we have two components that are described by two processes. Their behaviors are described by two extended timed automata shown in Figures 6 and 7. Figure 6 shows the timed automaton describing the behavior of the process representing the control of the water level. The states (in Location_1 and Location_2) are used to resume the execution of the tasks *fill()* or *empty()*. So, they are marked as committed, which means that they are being left directly after entering. The execution of the tasks *fill()* or *empty()* will take a

time specified by their computation time as defined in Table 1. The execution of the tasks due to their capacity (computation time) will finish in the following locations. The other locations are used to execute the behavior of the component as defined by the state space model when transitions are fired. The same thing is for the component controlling the temperature level (see Figure 7). These two processes have local variables (as x and y , F , and TJ) and shared variables (as V and T).

FIGURE 6: water level process

FIGURE 7: temperature level process

Each task has a set of parameters that should be defined. A task type: periodic, sporadic, or controlled. A task is described by the task code, its (worst-case) computation time and (relative) deadline, and if applicable optional parameters for priority (for fixed priority scheduling), period (for periodic tasks), and minimal inter-arrival time (for sporadic tasks). It is also possible to specify precedence constraints on the tasks using an editor for AND/OR precedence graphs, and resource access patterns using semaphores.

A task has also an interface. After a task has completed its execution it may update some of the system variables. In order to reflect that in the model, an interface between the task and the control structure is defined. An interface is an expression assigned to every task describing how the global variables are changed by the end of its execution.

In order to keep a shared resource by tasks in a consistent state only one party at a time is allowed to access it. To achieve this, a mutual exclusion mechanism between tasks is implemented based on the semaphores associated with the shared resources. The lock and unlock primitives, which delimit a critical section, are defined by the time points relative to the task execution time.

Sometimes tasks have to respect certain precedence relations to express, for example, explicit input/output relation between them. A precedence graph allows us to describe these relations inducing a partial order over a task set. In our example, four tasks are defined as follows.

TABLE 1: Tasks specification

Name	Behavior	Priority	Computing Time (C)	Deadline (D)	Period (P)	Offset	Max # of tasks	Interface
heat()	Controlled	1	6	12	X	X	1	TJ:=50
cool()	Controlled	4	8	12	X	X	1	TJ:=0
Fill()	Controlled	3	2	10	X	X	1	F:=5
empty()	Controlled	2	4	10	X	X	1	F:=0

Until now we have modeled the plant and the real-time application (the tasks) controlling the plant. The execution of the application (the tasks) is controlled by a real-time operating system (a real-time kernel). The important element of it is the scheduler. A semantic state of timed automata extended with tasks [4] is a triple $\langle l, v, q \rangle$, where l is the current control location, v denotes the current values of clocks and data variables, and q is the current task queue keeping all the released tasks to be executed. The semantics of an automaton is defined by a transition system in which the transition rules are parameterized by a scheduling policy to schedule the task queue when new tasks are released. Given an extended automaton and a scheduling policy, the related schedulability analysis problem is to check whether there exists a reachable state $\langle l, v, q \rangle$ of the automaton where the task queue q contains a task which misses its given deadline. Such states are called non-schedulable states. An automaton is said to be non-schedulable with the given scheduling policy if it may reach a non-schedulable state. Otherwise the automaton is schedulable. As the number of reachable states of an extended automaton is infinite, it is not obvious that the schedulability analysis problem is decidable.

Consider an automaton A and a scheduling strategy Sch . To check if A is schedulable with Sch , a timed automata $E(Sch)$ (the scheduler) and $E(A)$ (the task arrival pattern) are

constructed, and check the reachability of a predefined error state in the product automaton of the two. If the error state is reachable, automaton A is not schedulable with Sch .

To construct $E(A)$, the automaton A is annotated with distinct synchronization actions $release_i$ on all edges leading to locations labeled with the task name T_i (assume that only one task is annotated). The actions will allow the scheduler to observe when a task is released by A for execution. The behavior of the scheduler is described by the following extended timed automaton (Figure 8).

FIGURE 8: Timed automaton of the scheduler

The main idea is to keep track of the task queue, denoted by q on each step of the reachability analysis. Therefore in the encoding $E(Sch)$ there is a transition with the guard $nonschedulabe(q)$ from every location, where the queue is not empty (i.e. from all locations except Idle) to the error state. In the encoding, the task queue q is represented as a vector containing pairs of clocks (c_i, d_i) for every released task instance, called execution time and deadline clock respectively. The intuitive interpretation of the locations in $E(Sch)$ is as follows:

- Idle - the task queue is empty,
- Arrived(T_i) - the task instance T_i has arrived,
- Run(T_j) - the task instance T_j is running,
- Finished - a task instance has finished,
- Error - the task queue is non-schedulable.

Locations Arrived(T_i) and Finished are marked as committed, which means that they are being left directly after entering. The predicate $nonschedulabe(q)$ is used to denote the situation when the task queue becomes non-schedulable and naturally there is a transition labeled with the predicate leading to the error state. The predicate is encoded as follows.

$$\exists T_i \in q \text{ such that } d_i > D_i \quad (7)$$

d_i is the current time and D_i is the deadline of the task T_i . Sch is used in the encoding as a name holder for a scheduling policy to sort the tasks queue. A given scheduling policy is represented by the predicate $T_i := Hd(Sch(q))$. This means the task first scheduled with the policy Sch from the queue q . For example, Sch can be:

1. Highest Priority First (HPS) ($T_i \in q, \forall T_k \in q : \Pr(T_i) \geq \Pr(T_k)$).
2. Earliest Deadline First (EDF) ($T_i \in q, \forall T_k \in q : D_i - d_i \leq D_k - d_k$).
3. Least Laxity First (LLF) ($T_i \in q, \forall T_k \in q : c_i - d_i + D_i - C_i \leq c_k - d_k + D_k - C_k$).

C_i and c_i are computation and consumed times by the task T_i , respectively. Thus, the example timed automaton which is the composition of the two timed automata representing the behavior of the processes of water level changes and temperature level changes will be composed with this scheduler timed automaton using actions of releasing tasks and tasks properties defined in

Table 1. This product timed automaton is the overall specification of our embedded system that can be used for simulation and verification of properties including scheduling properties.

3. SIMULATION AND VERIFICATION

The simulation can be used to visualize the dynamic behavior of a system model as message sequence charts (Figure 9) and Gantt charts (Figure 10) [14]. It can be used to randomly generate possible execution traces, or alternatively the user can control the execution by selecting the transitions to be taken. The simulation can also be used to visualize error traces produced in the verification phase.

The simulation can be used to validate the dynamic behavior of the system and see how the tasks execute according to the task parameters and a given scheduling policy. The simulation shows a graphical representation of the generated trace showing the time points when the tasks are released, invoked, suspended, resumed, and completed.

FIGURE 9: Message sequence chart

FIGURE10: Gantt chart

3.1 Schedulability and Properties Verification

The schedulability verification is used to check that the tasks associated to a system model are guaranteed to always meet their deadline. In case schedulability analysis finds a task that may fail to meet its deadline, a trace is generated and visualized in the simulator. It is also possible to compute the worst-case response times of individual tasks. We have used the tool “TIMES” to check that our model is schedulable with fixed priority scheduling in the sense that all released tasks are guaranteed to meet their deadlines in all possible runs of the system. The property checked is

$$\forall W \neg (SCHEDULER.ERROR) \quad (8)$$

In addition, we have checked that the ready queue of the system is guaranteed to be *bounded* to two, meaning that the number of simultaneously released tasks will never be more than two.

$$\forall W (SCHEDULER.N0 + SCHEDULER.N1 + SCHEDULER.N2 + SCHEDULER.N3) \leq 2 \quad (9)$$

We have also checked that for each task type, the bound is one, i.e. there is never more than one task instance of each type released simultaneously

$$\forall W (SCHEDULER.N0 \leq 1) \wedge (SCHEDULER.N1 \leq 1) \wedge (SCHEDULER.N2 \leq 1) \wedge (SCHEDULER.N3 \leq 1) \quad (10)$$

A number of other correctness properties of our heating system behavior have also been checked, e.g. that the water level is always bounded within the interval [50, 250]. The same thing is checked for its temperature within the bounds [10, 100].

$$\forall W (V \leq 250 \wedge V \geq 50 \wedge T \leq 100 \wedge T \geq 10) \quad (11)$$

During the analysis we found and corrected several problems in the model. During the debugging, we often found the simulator of “TIMES” very useful. In particular the Gantt chart proved useful for tracing the executions of the tasks. In the Gantt chart view it is illustrated how tasks are executed according to the chosen scheduling policy. The message sequence chart is also useful for checking the tasks releases according to the processes behaviors.

4. CONCLUSION

We have presented an approach to analyze real-time embedded systems. This approach is based on modeling the embedded systems (in particular the scheduler) using the timed automata extended to specify scheduling. The environment (the plant and the real-time application) is modeled also by timed automata extended to specify the tasks. The plant (the controlled physical part) is usually complex. To overcome this problem, we have proposed to

decompose it to components that will be each specified by an independent extended timed automaton. The overall parallel composition of these extended timed automata will be the specification of the environment. We have used the tool “TIMES” to demonstrate our approach. This tool will compose all these extended automata before verifying the schedulability and system properties using its integrated model checking tool.

REFERENCES

- [1] Y. Abdeddaim, “Scheduling with timed Automata”, PhD thesis, Verimag, 2002.
- [2] R. Alur and D.L. Dill. “A Theory of Timed Automata”, *Theoretical Computer Science*, 126(2):183–235, April 1994.
- [3] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson and W. Yi, “TIMES: a Tool for Schedulability Analysis and Code Generation of Real-Time Systems”, In Proc. of FORMATS’03, LNCS, 2003.
- [4] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. TIMES - a tool for modelling and implementation of embedded systems. In Proc. of TACAS’02, volume 2280 of LNCS, pages 460_464. Springer, 2002.
- [5] M. Anoop, “Security needs in embedded systems”, 2008.
- [6] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-vincentelli. Scheduling for embedded real-time systems. *IEEE Design & Test of Computers*, 15(1):71_82, 1998.
- [7] M. Barr, “Embedded Systems Glossary”, 2007.
- [8] M. Bourahla, “Verification of complex real-time systems using rewriting logic”, *Journal of Computing and Information Technology*, volume 17(3), 2009.
- [9] J. Catsoulis, “Designing Embedded Hardware”, O’Reilly, 2005.
- [10] E. Fersman, L. Mokrushin, P. Petterson, and W. Yi, “schedulability analysis using two clocks”, in Proc. TACAS 2003, volume LNCS 2619, pages 224-239, 2003.
- [11] M. Krichen and S. Tripakis. Black-box Conformance Testing for Real-Time Systems. In *Model Checking Software: 11th International SPIN Workshop*, volume LNCS 2989. Springer, April 2004.
- [12] K. Larsen, Marius Mikucionis, Brian Nielsen and Arne Skou, “Testing RealTime Embedded Software using UPPAAL-TRON: An Industrial Case Study”, *EMSOFT’05*, 2005
- [13] K. Larsen, P. Pettersson, and W. Yi. “Uppaal in a Nutshell”, *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, 1997.
- [14] The TIMES Tool - user manual, url:<http://www.timestool.com>
- [15] L. Waszniowski and Z. Hanzalek, “Analysis of real time operating system based applications. In Proc. FORMATS’03, 2003.