TIMES - A Tool for Modelling and Implementation of Embedded Systems

Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi*

Uppsala University, Sweden.
E-mail: {tobiasa,elenaf,leom,paupet,yi}@docs.uu.se.

1 Introduction

TIMES is a modelling and schedulability analysis tool for embedded real-time systems, developed at Uppsala University in 2001. 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.

The main features of TIMES are:

- A graphical editor for timed automata extended with tasks [FPY02], which allows the user to model a system and the abstract behaviour of its environment In addition the user may specify a set of preemptive or non-preemtive tasks with parameters such as (relative) deadline, execution time, priority, etc.
- A simulator, in which the user can validate the dynamic behaviour of the system and see how the tasks execute according to the task parameters and a given scheduling policy. The simulator shows a graphical representation of the generated trace showing the time points when the tasks are released, invoked, suspended, resumed, and completed.
- A verifier for schedulability analysis, which is used to check if all reachable states of the complete system are schedulable that is, all task instances meet their deadlines. A symbolic algorithm has been developed based on the DBM techniques and implemented based on the verifier of the UPPAAL tool [LPY97].
- A code generator for automatic synthesis of C-code on LegoOS platform from the model. If the automata model is schedulable according to the schedulability analyser the execution of the generated code will meet all the timing constraints specified in the model and the tasks.

An screen-shot of the TIMES tool is shown in Fig. 1. In section 3 we describe the tool and its main functionalities in more details. The modelling language and the theoretical foundation of TIMES is based on the model of timed automata with tasks, described in the following section.

* Corresponding author: Wang Yi, Department of Information Technology, Uppsala University, Box 325, 751 05, Uppsala, Sweden. Email: yi@docs.uu.se

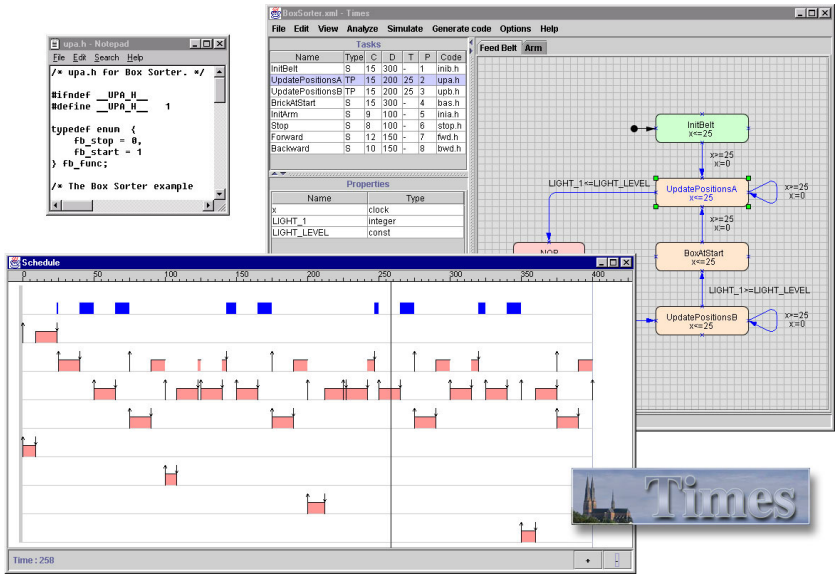

Fig. 1. Screen-shot of the TIMES tool.

2 Input Language

The core of the input language of TIMES is timed automata with real time tasks (TAT), described in details in [FPY02] included in this volume, with one major addition that shared variables between automata and tasks are allowed. Here we give a brief introduction to the model. A TAT is a timed automaton extended with tasks triggered by events. A task is an executable program (written in a programming language e.g. C) characterized by its worst execution time and deadline, and possibly other parameters such as priorities etc. for scheduling. A task may update a set of variables using assignments in the form $x := E$ where x is a variable and E is an expression (computed by the task and the value of E is returned when the task is finished). The variables may also be changed and tested by an automaton. Intuitively an edge leading to a location in the automaton denotes an event triggering the task, and the guard (clock constraints) on the transition specifies the possible arrival times of the event. This allows us to describe concurrency and synchronization, and real time tasks which may be periodic, sporadic, preemptive and (or) non-preemptive, with or without precedence constraints. An automaton is schedulable if there exists a (preemptive or non-preemptive) scheduling strategy such that all possible sequences of events accepted by the automaton are schedulable in the sense that all associated tasks can be computed within their deadlines.

Semantically, an extended timed automaton may perform two types of transitions just as standard timed automata. But the difference is that delay transitions correspond to the execution of running tasks with highest priority (or earliest

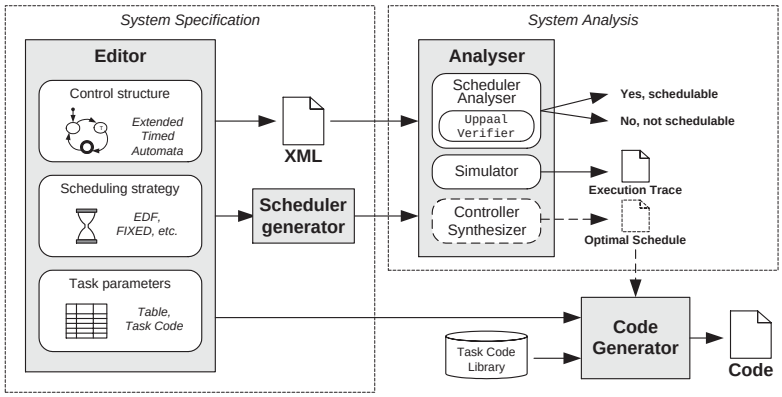

Fig. 2. Overview of the TIMES tool.

deadline) and idling for the other tasks waiting to run. Discrete transitions corresponds to the arrival of new task instances. Whenever a task is triggered, it will be put in a task queue for execution (corresponding to the ready queue in operating systems). Thus there may be a number of processes (released tasks) running logically in parallel during the execution of an automaton.

The scheduling problem of TAT is to verify that all released tasks are guaranteed to always meet their deadlines when executed according to a given scheduling policy. In TIMES the analysis is performed by transforming a TAT system into ordinary timed automata extended with subtraction operations on clocks, and encoding the schedulability problem to a reachability problem as described in [FPY02].

3 Tool Overview

An overview of the TIMES tool is shown in Fig. 2. The tool is divided in three parts: a system specification part, a system analysis part, and a code generator[1].

In the system specification part, the user models a system to be analysed. A system specification in TIMES consists of three parts: the control automata modelled as a network of timed automata extended with tasks [FPY02], a task table with information about the processes triggered (released) when the control automata changes location, and a scheduling policy.

The System Editor tool, shown in Fig. 1, is a an editor for drawing the control automata of the system model. It also provides a table for defining the task parameters. The task parameters currently supported are: (relative) deadline, execution time, period, priority, a reference to the task code, and a field indicating the task behaviour. A task has one of the following three behaviours: "S"

[1] The components indicated with dashed lines in Fig. 2 are planed extension not yet included.

for sporadic, "TP" for temporarily periodic, and "P" for periodic. The currently supported scheduling polices are: first-come first-served, fixed priority (i.e. according to the priorities assigned in the task table), rate monotonic, deadline monotonic, and earliest-deadline first. All polices can be either preemptive or non-preemptive.

The output of the System Editor is an XML representation of the control automata. The information from the task table and the scheduling policy, are used by the Scheduler Generator to generate a scheduler automaton that is composed in parallel with the controller automata to ensure that the system behaves according to the scheduling policy and the task parameters. If the scheduling policy is non-preemptive, the scheduler automaton is an ordinary timed automaton. If the scheduling policy is preemptive, the scheduler automaton is modelled as a variant of timed automata in which clock variables may be updated by subtractions. For more information about how to solve scheduling problems of timed automata with tasks using timed automata, see [FPY02].

The parallel composition of the control automata and the scheduler automaton is used as input to the System Analyser that consist of two main components: a Simulator, and a Schedule Analyser. In the Simulator the user may debug the system by exploring the dynamic behaviour of the system model and observe how the tasks execute according to the chosen scheduling policy. Screen-shots of the Simulator are shown in Fig. 1 and 4. As shown, the Simulator displays a diagram with $n + 1$ lines, where n is the number of tasks. On the upper line, it is indicated in blue (or black) when no task is executing. The lower n lines are associated with one task each, and used to show in red (or gray) when the corresponding task is executing. As time progresses the diagram grows to the right (i.e. the time line goes from left to right). Note that at any moment in time either, one or zero tasks are executing, or a switch takes place.

From the System Analyser it is also possible to invoke the Schedule Analyser that performs schedulability analysis of the system. The analysis is performed by rephrasing scheduling to a reachability problem that is solved with an extended version of the verifier of the UPPAAL tool [LPY97]. In case the output of the analysis is negative, the analyser generates a trace of the system that ends in a state in which one of the system task fails to meet its deadline.

The Code Generator of TIMES uses the control automata and the task programs to synthesise executable C-code. Currently the only supported platform is the LegOS operating system. Support for other platforms will be included in future versions.

4 Example: Box Sorter

In this section, we describe how TIMES is applied to the box sorter example of [LPY97]. The system sorts red and black boxes arriving on a belt by kicking the black boxes of the belt. The physical components are the feed belt, a light sensor, and a kick-off arm. The programs controlling the feed belt and the arm are modelled using two timed automata with tasks, as shown in Fig. 3.

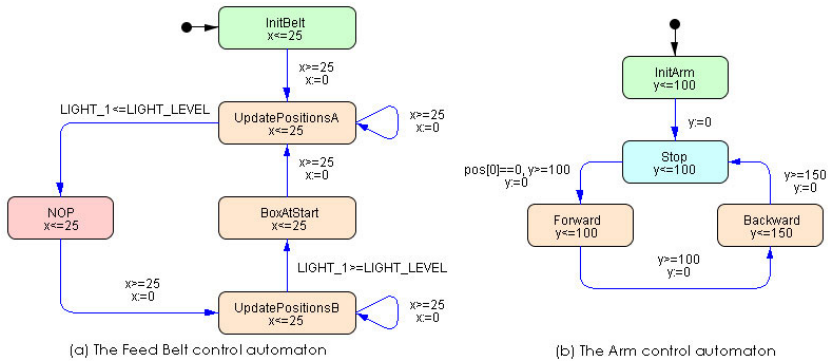

Fig. 3. The two Control Automata of the Box Sorter.

The tasks of the automaton in Fig. 3(a) uses a global array named pos to store the positions of the black boxes on the belt. The automaton in Fig. 3(b) controls the kick-off arm.

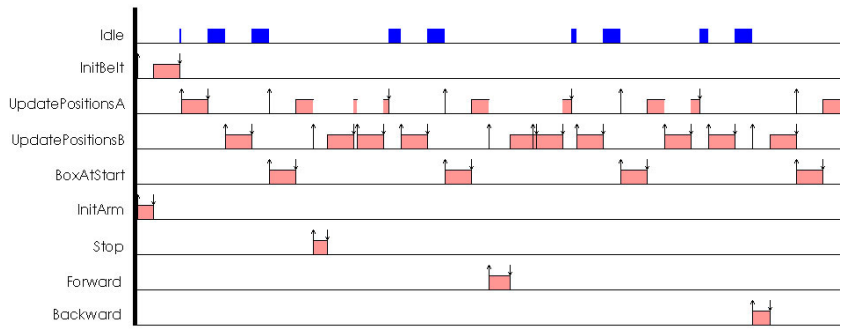

Fig. 4. The Box Sorter Schedule.

Fig. 4 shows an execution trace of the box sorter system, as it appears in the simulator window of the TIMES tool. Note how the simulator indicates that e.g. the second invocation of task UpdatePositionsA is delayed and preempted twice by higher priority tasks.

References

1. Elena Fersman, Paul Pettersson, and Wang Yi. Timed Automata with Asynchronous Processes: Schedulability and Decidability. In *Proc. of the 8*th *Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2002.
2. Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.