TIMES: A Tool for Schedulability Analysis and Code Generation of Real-Time Systems

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

Department of Information Technology
Uppsala University, P.O. Box 337, SE-751 05 Uppsala, Sweden
{tobiasa,elenaf,leom,paupet,yi}@it.uu.se

Abstract. TIMES is a tool suite designed mainly for symbolic schedulability analysis and synthesis of executable code with predictable behaviours for real-time systems. Given a system design model consisting of (1) a set of application tasks whose executions may be required to meet mixed timing, precedence, and resource constraints, (2) a network of timed automata describing the task arrival patterns and (3) a preemptive or non-preemptive scheduling policy, TIMES will generate a scheduler, and calculate the worst case response times for the tasks. The design model may be further validated using a model checker e.g. UPPAAL and then compiled to executable C-code using the TIMES compiler. In this paper, we present the design and main features of TIMES including a summary of theoretical results behind the tool. TIMES can be downloaded at www.timestool.com.

1 Introduction

In classic scheduling theory, real time tasks (processes) are usually assumed to be periodic, i.e. tasks arrive (and will be computed) with fixed rates periodically. Analysis based on such a model of computation often yields pessimistic results. To relax the stringent constraints on task arrival times, we have proposed to use automata with timing constraints to model task arrival patterns [1]. This yields a generic task model for real time systems. The model is expressive enough to describe concurrency and synchronization, and real time tasks which may be periodic, sporadic, preemptive or non-preemptive, as well as precedence and resource constraints. We believe that the model may serve as a bridge between scheduling theory and automata-theoretic approaches to system modeling and analysis. The standard notion of schedulability is naturally generalized to automata. An automaton is schedulable if there exists a 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. It has been shown that the schedulability checking problem for such models is decidable [1]. A recent work [6] shows that for fixed priority scheduling strategy,

* Corresponding author.

K.G. Larsen and P. Niebert (Eds.): FORMATS 2003, LNCS 2791, pp. 60–72, 2004.

the problem can be efficiently solved by reachability analysis on timed automata using only 2 extra clock variables. The analysis can be done in a similar manner to response time analysis in classic Rate-Monotonic Scheduling.

The first main function of TIMES is developed based on these recent results on schedulability analysis. Its second main function is code generation. Code generation is to transform a validated design model to executable code whose execution preserves the behaviour of the model. Given a system design model in TIMES including a set of application tasks, task constraints, tasks arrival patterns and a scheduling policy adopted on the target platform, TIMES will generate a scheduler and calculate the worst-case response times for all tasks. The model may be further validated by a model-checker e.g. UPPAAL [9], and then compiled to executable C-code. We assume that the generated code will be executed on a platform on which every annotated task in the design model will not take more than the given computing time. Further assume that the platform guarantees the synchronous hypothesis in the sense that the times for handling system functions e.g. collecting external events can be ignored compared with the computing times and deadlines for the annotated tasks. Under these assumptions on the platform, code generation is essentially to resolve non-determinism in the design model. In TIMES, time non-determinism is resolved by the maximal progress assumption, that is, whenever a transition is enabled, it should be taken. External non-determinism in accepting events is resolved using priority order.

The rest of the paper is organized as follows: the next section describes the core of the input TIMES language and its informal semantics. Section 3 summarizes briefly the main theoretical work on schedulability analysis and code synthesis. Section 4 describes the main features of TIMES, the tool architecture and the main components in the implementation. Section 5 concludes the paper with a summary of ongoing work and future development.

2 Task Models in TIMES

The two central concepts in TIMES are *task* and *task model*. A task (or task type) is an executable program (e.g. in C) with task parameters: worst case execution time and deadline. A task may have different *task instances* that are copies of the same program with different inputs. A task model is a task arrival pattern such as periodic and sporadic tasks. In TIMES, timed automata are used to describe task arrival patterns.

2.1 Tasks Parameters and Constraints

Following the literature [4], we consider three types of task constraints.

Timing Constraints. A typical timing constraint on a task is deadline, i.e. the time point before which the task should complete its execution. We assume that the *worst case execution times* (WCET) of tasks are known (or pre-specified). We characterize a task as a pair of natural numbers denoted (C, D) with $C \leq D$,