

# Software Synthesis Guaranteeing Timing Constraints

Tobias Amnell    Elena Fersman    Paul Pettersson    Wang Yi

Department of Computer Systems, Information Technology,  
Uppsala University, Box 325, 751 05 Uppsala, Sweden,  
Web: <http://www.docs.uu.se/docs/rtmv/>,  
E-mail: {tobiasa,elenaf,paupet,yi}@docs.uu.se.

**Abstract** We present recent work of the “UPPAAL-group” at the Department of Computer Systems of Uppsala University. The work aims at developing a common framework for automatic synthesis of executable code with guaranteed timing constraints, from high level design models.

## Motivation

A key facility of many commercial tools for development of embedded software is to automatically synthesize fully executable code (i.e. code generation) from design models. However, few of the existing tools are capable of producing software with *predictable* timing behavior, i.e. code which is known a priori to satisfy given timing constraints. In fact, most of the commercial tools trust the software developer to resolve the timing issues that arises when the generated code should be executed on a specific target platform, such as e.g. analysing if all tasks will meet their deadlines.

In contrast, research tools for real-time systems such as UPPAAL [4] and Kronos [2] are often dedicated to analysis and abstraction of formal high-level design description. This has proven useful for finding errors and checking correctness properties in many case studies, e.g. [5]. However, it provides little or no support for producing the actual program code to be executed in the final implementation.

In this work, we combine results and ideas from model-checking, scheduling, and synchronous programming to develop a framework for the development of real-time embedded systems. The goal is to support, on one hand formal specification, valida-

tion, analysis design, and on the other code synthesis that guarantees that certain timing constraints are met when executed on the target system.

## Code Synthesis

We propose a way of synthesising code from timed automata models, which has a predictable timing behavior. Inspired by the design philosophy of synchronous languages e.g. Esterel [1], we assume that the underlying real-time operating system guarantees the *synchrony hypothesis*<sup>1</sup>. We extend the model of timed automata by associating with each node a task (or several tasks in the general case). A task is assumed to be an executable program with two given parameters: its worst case execution time and its deadline. An example of the resulting model, called *executable timed automata*, is shown in Figure 1.

Intuitively, a discrete transition in an executable timed automaton denotes an event releasing a task and the clock constraints (guard) on the transition specifies all the possible arrival times of the associated task. When the task is released it is inserted into the ready queue of the operating system. Note that in the simple automaton shown in Figure 1, an instance of task A could be released before the preceding instance of task P has been computed. This means that the scheduling queue may contain at least P and A. In fact, instances of all four tasks may appear in the queue at the same time.

---

<sup>1</sup>That is, the operating system calls take little time compared to the execution times and deadlines of the tasks.

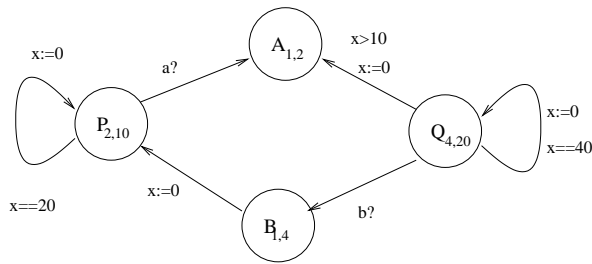


Figure 1: A system with 4 tasks, one associated with each node. The tasks P and Q are periodic with periods 20 and 40 respectively (specified by the constraints:  $x:=20$  and  $x:=40$ ). The tasks A and B are sporadic or event driven (by the events a and b respectively). The pair of numbers in each node give the computation times and deadlines of the task respectively.

**Schedulability Analysis** In the model of executable timed automata, tasks in the ready queue are executed according to a chosen scheduling strategy, e.g. earliest deadline first. In [3], it is shown that the schedulability problem can be checked by reachability analysis for non-preemptive tasks. The problem is equivalent to showing that all reachable states of the automata are schedulable. We are working on extending this result to more general execution models (e.g. preemptive tasks) and other types of analysis (e.g. maximum length of the ready queue).

**Synthesizing Code for Controllers** Timed automata annotated with tasks, as described above, are used as a design model of a control program. The discrete transitions (i.e. the control structure) of the automaton and the associated tasks are implemented using a small set of (common) system calls and assumes that light-weight threads are provided by the underlying real-time operating system. This enables code synthesis for a variety of different target platforms. As a result, if an automaton is schedulable and the synchrony hypothesis is guaranteed by the underlying operating system, the generated code will when executed meet the constraints (timed and other) imposed on the tasks.

We have implemented a prototype that synthesizes C-code for the *legOS* operating system. *legOS* runs on the LEGO Mindstorm control brick that is equipped with an 8-bit Hitachi micro-controller.

This hardware is rather typical for embedded systems of the kind we are targeting. Our prototype has so far given us promising evidence that our approach is viable.

**A Small Example** The implemented code generation has been used to generate a control program for the conveyor belt shown in Figure 2. A light sensor detects the color of the bricks on the belt, and a program controls the arm at the end of the belt so that all black bricks are being kicked off the belt, whereas all red bricks reaches the end of the belt.

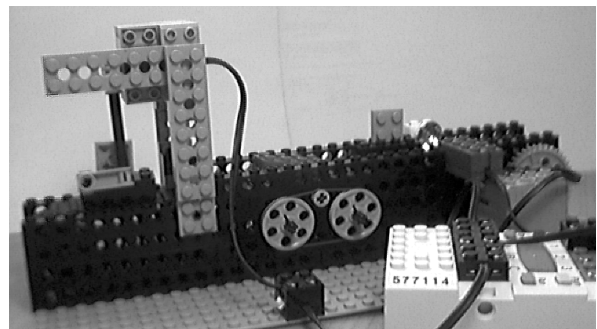


Figure 2: A conveyor belt built in LEGO.

## References

- [1] G. Berry and G. Gonthier. The Synchronous Programming Language ESTEREL: Design, Semantics, Implementation. *Science of Computer Programming*, 19:87–152, 1992.
- [2] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A Model-Checking Tool for Real-Time Systems. In *Proc. of the 10th Int. Conf. on Computer Aided Verification*, number 1427 in Lecture Notes in Computer Science, pages 546–550. Springer-Verlag, 1998.
- [3] Christer Ericsson, Anders Wall, and Wang Yi. Timed Automata as Task Models for Event-Driven Systems. *Proceedings of RTSCA '99*, 1999.
- [4] Kim G. Larsen, Paul Pettersson, and Wang Yi. UP-PAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, October 1997.
- [5] Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal Design and Analysis of a Gear-Box Controller. In *Proc. of the 4th Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 281–297. Springer-Verlag, March 1998.