UPPAAL – Present and Future[1]

Gerd Behrmann, Kim G. Larsen
Basic Research in Computer Science, Aalborg University, Denmark

Oliver Möller
Basic Research in Computer Science, Aarhus University, Denmark

Alexandre David, Paul Pettersson, Wang Yi
Department of Information Technology, Uppsala University, Sweden

Abstract

UPPAAL is a tool for modelling, simulation and verification of real-time systems, developed jointly by BRICS at Aalborg University and the Department of Computer Systems at Uppsala University. The tool is appropriate for systems that can be modelled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels or shared variables. Typical application areas include real-time controllers and communication protocols in particular, those where timing aspects are critical. In this paper, we review the status of the currently distributed version of the tool as well as facilities to be found in upcoming releases.

1 Current Version of UPPAAL

1.1 Background

UPPAAL [LPY97] consinsts of three main parts: a description language, a simulator and a model checker. The description language is a non-deterministic guarded command language with real-valued clock variables and simple data types. It serves as a modelling or design language to describe system behavior as networks of automata extended with clock and data variables. The simulator is a validation tool which enables examination of possible dynamic executions of a system during early design (or modelling) stages and thus provides an inexpensive mean of fault detection prior to verification by the model checker which covers the exhaustive dynamic behaviour of the system. The model checker is to check invariant and bounded-liveness properties by exploring the symbolic state-space of a system, i.e., reachability analysis in terms of symbolic states represented by constraints.

Since the first release of UPPAAL in 1995, the tool has been further developed by the teams in Aalborg and Uppsala. Figure 1 illustrates how this has affected the performance of the tool in terms of three examples from the literature. The diagram shows how the space and time requirements of UPPAAL improved in the period December 1996 to September 1998 when compiled with the same compiler and executed on the same machine. The time reduction is similar [Pet99].

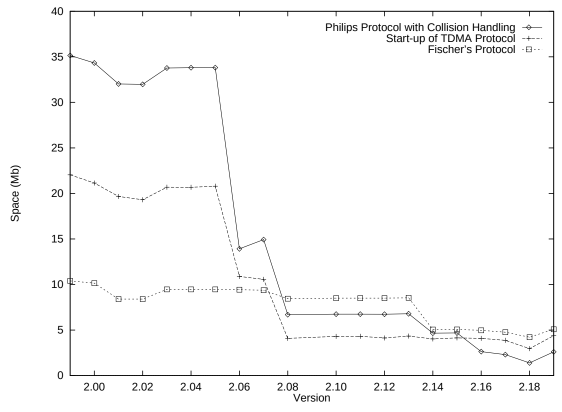


Figure 1: Space (in Mb) benchmarks for UPPAAL version 1.99 to 2.19. Version 1.99 and 2.19 are dated December 1996 and September 1998 respectively. All tool versions were compiled with gcc 2.7.2.3 and executed on the same Pentium II 375 MHz machine.

In July 1999 a new version of UPPAAL, called UPPAAL2k, was released. The new version, which required almost two years of development, is designed to improve the graphical interface of the tool, to allow for easier maintenance, and to be portable to the most common operating systems while still preserving UPPAAL's ease-of-use and efficiency. To meet these requirements the new version is designed as a client/server application with a verification server providing efficient C++ services to a Java client over a socket based protocol. This design also makes it possible to execute the server and the GUI on two different machines. UPPAAL2k is currently available for Linux, SunOS and MS Windows platforms.

[1]This work is partially supported by the European Community Esprit-LTR Project 26270 VHS (Verification of Hybrid systems), and the AIT-WOODDES Project No IST-1999-10069.

It can be downloaded from the Uppaal home page `http://www.uppaal.com/`. Since July 1999, the tool has been downloaded by more than 800 different users in 60 countries.

1.2 GUI

The new GUI, shown in Figure 2, has new interfaces for the three main tool components of Uppaal, i.e., the system editor, the simulator and the verifier. Being integrated in one common interface, the three tools now have more uniform interfaces compared to previous Uppaal versions. The three tools operate on the same internal system model which makes exchange of information between the tools easier, e.g., loading a diagnostic trace generated by the verifier into the simulator for further inspection. In addition, several new functionalities have been implemented in the tool. For example, the new system editor has been tailored and extended for the new system description language of Uppaal2k (see below), the simulator has been modified to allow the user to configure the level of details to be displayed of the simulated system, and the verification interface has been enriched with a requirement specification editor which stores the previous verification results of a logical property until the property or the system description is modified.

The new Uppaal version also has a richer modelling language than its predecessors. The new language supports process templates and more complex (bounded) data structures, such as data variables, constants, arrays etc. A process template in the new language is a timed automaton extended with a list of formal parameters and a set of locally declared clocks, variables and constants. Typically, a system description will consist of a set of instances of timed automata declared from the process templates, and of some global data, such as global clocks, variables, synchronisation channels etc. In addition, automata instances may also be defined from templates re-used from existing system descriptions. Thus, the adopted notion of process templates (particularly when used in combination with the possibility to declare local process data) allows for convenient re-use of existing models.

1.3 Verifier

A main focus of the Uppaal project is to develop efficient algorithms and data structures for the verification of timed systems. The new verification server of Uppaal2k contains some recent developments in this area (though some of the implementations are not yet available in the public version).

In two recent papers [BLP$^+$99, LWYP99], Behrmann et al presents a new data structure called *Clock Difference Diagrams*, CDDs. The new structure is BDD-like (it allows for sharing of isomorphic sub trees) but intended for representing and efficiently manipulating the non convex subsets of the Euclidean space encountered during verification of timed automata. The CDDs have been implemented in Uppaal to perform the symbolic state-space exploration instead of the normally used data structure, called DBMs. In an experiment where the tool was applied to eight industrial examples, the space savings using CDDs were between 46% and 99% with moderate increase in run time.

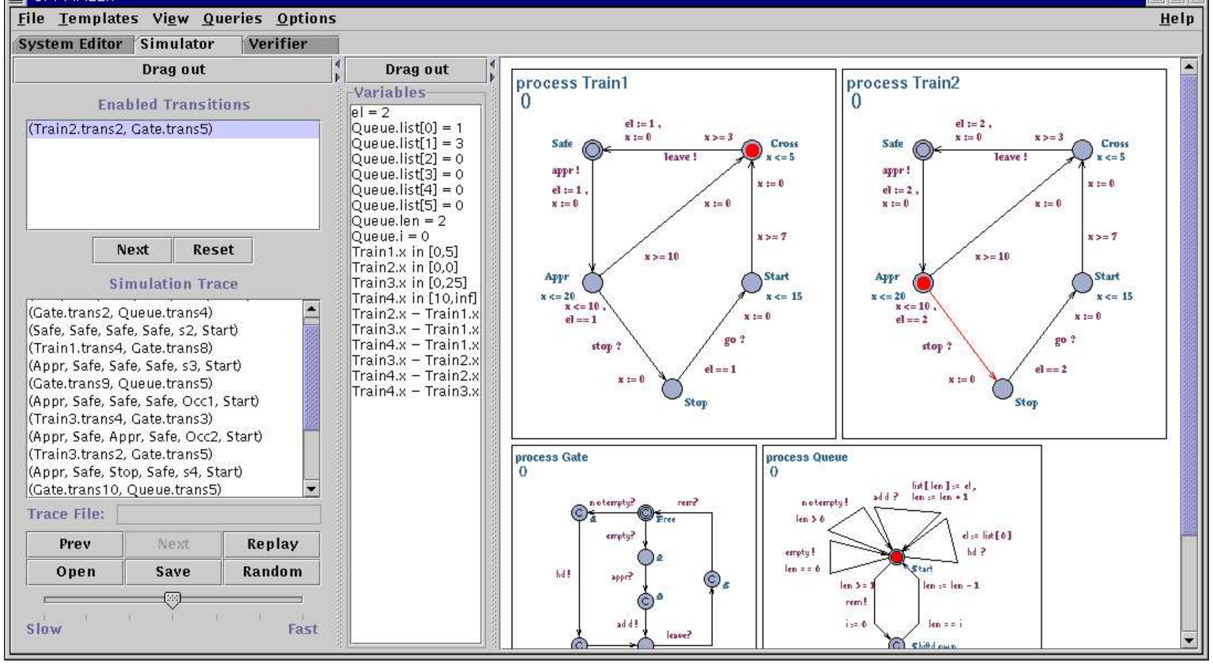
Another paper [LNAB$^+$98] describes a new verification technique called *Compositional Backwards Reachability*, CBR. The technique uses *compositionality* and *dependency analysis* to improve the efficiency of symbolic model checking of state/event models. In an untimed setting, the technique has made possible automatic verification of very large industrial design. For example a system with 1421 concurrent machines was checked in less than 20 minutes on a standard PC. An implementation of this technique for timed systems is currently under development and has already proved its applicability on some benchmark examples.

The Uppaal2k verification server has also been extended with some verification techniques described elsewhere in the literature. The current version supports the *bit-state hashing* under-approximation technique which has been successfully used in the model-checking tool SPIN for several years. A technique for generating an over-approximation of a system's reachable state-space based on a *convex-hull* representations of constraints is also supported. Finally, an abstraction technique based on *(in-)active clock reductions* is available.

1.4 Case Studies

Uppaal2k has been applied in several case studies. In this section we briefly describe some of the major and more recent case studies performed.

In an ongoing case study [AJ01], Uppaal is applied to model and analyze a generalized version of a car locking system developed by Saab Automobile. The locking system is distributed over several nodes in the internal communication network that exists in all modern vehicles. The system consists of a central node gathering information and based on this instructing sub nodes attached to the physical hardware to lock or unlock doors, trunk lid, etc. The input sources are different kinds of remote controllers, speed sensors, automatic re-locking timeouts etc., which based on predefined rules may activate the locking mechanism. The model of the system is derived from the actual functional requirements of the locking system used at Saab Automobile. During the currently ongoing work with verifying the functional requirements of the model, some inconsistencies and other problems between requirements have been found and pointed out to the engineers.



```
(Train2.trans2, Gate.trans5)
```

```
(Gate.trans2, Queue.trans4)
(Safe, Safe, Safe, Safe, s2, Start)
(Train1.trans4, Gate.trans8)
(Appr, Safe, Safe, Safe, s3, Start)
(Gate.trans9, Queue.trans5)
(Appr, Safe, Safe, Safe, Occ1, Start)
(Train3.trans4, Gate.trans3)
(Appr, Safe, Appr, Safe, Occ2, Start)
(Train3.trans2, Gate.trans5)
(Appr, Safe, Stop, Safe, s4, Start)
(Gate.trans10, Queue.trans5)
```

```
el = 2
Queue.list[0] = 1
Queue.list[1] = 3
Queue.list[2] = 0
Queue.list[3] = 0
Queue.list[4] = 0
Queue.list[5] = 0
Queue.len = 2
Queue.i = 0
Train1.x in [0,5]
Train2.x in [0,0]
Train3.x in [0,25]
Train4.x in [10,inf]
Train2.x − Train1.x
Train3.x − Train1.x
Train4.x − Train1.x
Train3.x − Train2.x
Train4.x − Train2.x
Train4.x − Train3.x
```

Figure 2: UPPAAL2k on screen.

In [DY00], David and Wang report on an industrial application of UPPAAL to model and debug a commercial field bus communication protocol, AF100 (Advant Field-bus 100) developed and implemented by the process control industry for safety-critical applications. The protocol has been running in various industrial environments over the world for the past ten years. Due to the complexity of the protocol and various changes made over the years, it shows occasionally unexpected behaviours. During the case study, a number of imperfections in the protocol logic and its implementation are found and the error sources are debugged based on abstract models of the protocol; respective improvements have been suggested.

In [HLP00], Hune et al. addresses the problem of synthesising production schedules and control programs for the batch production plant model built in LEGO® MINDSTORMS™ RCX™. A timed automata model of the plant which faithfully reflects the level of abstraction needed to synthesise control programs is described. This makes the model very detailed and complicated for automatic analysis. To solve this problem a general way of adding guidance to a model by augmenting it with additional guidance variables and transition guards is presented. Applying the technique makes synthesis of control problems feasible for a plant producing as many as 60 batches. In comparison, only two batches could be scheduled without guides. The synthesized control programs have been executed in the plant. Doing this revealed some model errors. The papers [Hun99, IKL+00] also consider systems controlled by LEGO® RCX™ bricks. Here the studied problem is that of checking properties of the actual programs, rather than abstract models of programs. It is shown how UPPAAL models can be automatically synthesized from RCX™ programs, written in the programming language *Not Quite C*, NQC. Moreover, a protocol to facilitate the distribution of NQC programs over several RCX™ bricks is developed and proved to be correct. The developed translation and protocol are applied to a distributed LEGO® system with two RCX™ bricks pushing boxes between two conveyer belts moving in opposite directions. The system is modelled and some verification results with UPPAAL2k are reported.

In [KLPW99], Kristoffersen et. al. present an analysis of an experimental batch plant using UPPAAL2k. The plant is modelled as a network of timed automata where automata are used for modelling the physical components of the plant, such as the valves, pumps, tanks etc.

2 Recent Developments in UPPAAL

Several research activities are conducted within the context of UPPAAL. In particular, extensions of the tool to allow for parametric models [HRSV], probabilistic models and hybrid system models [CL00] have been or are under investigation. Also, the state-explosion problem, which is even more severe in the context of real-time, has been subject to substantial research. Beside the already described BDD-like datastructure CDD [BLP+99, LWYP99], real-time extensions of the partial order reduction technique have been suggested [BJLY98]. The research results most likely to be found in versions to be released shortly will be described in somewhat more detail in the following sections.

2.1 Distributed Uppaal

Real time model checking is a time and memory consuming task, quite often reaching the limits of both computers and the patience of users. An increasingly common solution to this situation is to use the combined power of normal computers connected in a cluster. Good results have recently been achieved for Uppaal by distributing both the model checking algorithm and the main data structures [BHV00].

At the core of Uppaal we find a state-space exploration algorithm. In principal, we might think of this as a variation of searching the states (nodes) of an oriented graph. For this, two data structures are responsible for the potentially huge memory consumption. The first – the Waiting list – contains the states that have been encountered by the algorithm, but have not been explored yet, i.e., the successors have not been determined. The second – the Passed list – contains all states that have been explored. The algorithm takes a state from the Waiting list, compares it with the Passed list, and in case it has not been explored, the state itself is added to the Passed list while the successors are added to the Waiting list.

The distributed version of this algorithm is similar. Each node (processing unit) in the cluster will hold fragments of both the Waiting list and the Passed list according to a distribution function mapping states to nodes. In the beginning, the distributed Waiting list will only hold the initial state. What ever node hosts this state will compare it to its still empty Passed list fragment and consequently explore it. Now, the successors are distributed according to the distribution function and put into the Waiting list fragment on the respective nodes. This process will be repeated, but now several nodes contain states in their fragment of the Waiting list and quickly all nodes become busy exploring their part of the state space. The algorithm terminates when all Waiting list fragments are empty and no states are in the process of being transfered between nodes.

The distribution function is in fact a hash function. It distributes states uniformly over its range and hence implements what is called *random load balancing*. Since states are equally likely to be mapped to any node, all nodes will receive approximately the same number of states and hence the load will be equally distributed.

This approach is very similar to the one taken by [SD97]. The difference is that Uppaal uses symbolic states, each covering (infinitely) many concrete states. In order to achieve optimal performance, the lookup performed on the Passed list is actually an inclusion check. An unexplored symbolic state taken from the Waiting list is compared with all the explored symbolic states on the Passed list, and only if none of those states cover (include) the unexplored symbolic state it is explored. For this to work in the distributed case, the distribution function needs to guarantee that potentially overlapping symbolic states are mapped to the same node in the cluster. A symbolic state can actually be divided into a discrete part and a continuous part. By only basing the distribution on the discrete part, the above is ensured.

One oddity is that, depending on the search order, building the complete reachable state-space can result in varying number of states being explored. Experiments have shown that breadth first order is close to optimal when building the complete reachable state-space. Unfortunately, ensuring strict breadth first order in a distributed setting requires synchronizing the nodes, which is undesirable. Instead, we order the states in each Waiting list fragment according to their distance from the initial state. his results in an approximation of the breadth first order. Experiments have shown that this order drastically reduces the number of explored states compared to simply using a FIFO order.

This version of Uppaal has been used on a Sun Enterprise 10000 with 24 CPUs and on a Linux Beowulf cluster with 10 nodes. Good speedups have been observed on both platforms when verifying large systems (around 80% of optimal at 23 CPUs on the Enterprise 10000).

2.2 Cost-Optimal Uppaal

Uppaal was initially intended to prove the correctness of a real time systems with respect to their specification. If a system does not meet the specification Uppaal finds an error state and can produce diagnostic information on how to reach this error state. However, we often prefer to think of these states as desired goal states and not as error states. If for example four persons have to cross a bridge that can only carry two persons at a time, one would like to know whether they can reach the safe side, given additional constraints and deadlines. This example extends to bigger systems from, e.g., the process industry. It is then often valuable to know whether it is possible to schedule the production steps such that all constraints are met. This approach was used in [Feh99, HLP00] to derive feasible schedules for a part of a steel plant in Ghent, Belgium, and a Lego model of this plant, respectively.

Even though it is often hard to find a solution, as soon as a feasible solution is found, one might wonder whether this is the optimal solution; whether no better solutions exist. To address this problem we included concepts that are well known from branch and bound algorithms to Uppaal. It is then possible to derive optimal traces for *Uniformly Priced Timed Automata* (UPTA) [BFH$^+$]. In this model the cost increases with a fixed rate as time elapses, or with a certain amount if a transition is taken. The cost is treated as a special clock with extra operations, but such that we can still use the efficient data structures currently used in Uppaal. First results for the steel plant and several benchmark problems were obtained in [BFH$^+$], and we hope to include in the next release of Uppaalan option for detecting optimal traces to goal states.

To be able to find time-optimal traces is very useful, but in many situations we would like to have a more general notion of cost. To be able to model for example machines that use a different amount of energy per time unit we proposed the model of *Linearly Priced Timed Automata* (LPTA). This model extends timed automata with *prices* on all transitions and locations. In these models, the cost of taking an action transition is the price associated with the transition, and the cost of delaying d time units in a location is $d \cdot p$, where p is the price associated with the location. The cost of a trace is simply the accumulated sum of costs of its delay and action transitions.

To deal with LPTA we introduce *priced zones*, which assign to a zone a linear function that defines the minimal cost of reaching a state in that zone. In [BFH$^+$00] it was shown that given a set of goal states the cost-optimal trace is computable. This result is quite remarkable since several similar extensions of timed automata have been proven to be undecidable. But furthermore we now even have a prototype implementation that allows us to perform the first experiments [LBB$^+$01].

From first attempts that use Uppaal to show schedulability for some notorious problems, we have now a general model that allows us to find a trace with the minimum cost of all traces ending in a set of goal states. In this approach, the automata based modeling languages of the verification tools serves as input language. These modeling languages are very well-suited in this respect, as they allow for easy and flexible modeling of systems consisting of several parallel components that interact in a time-critical manner and constrain the behavior of each other in a multitude of ways.

2.3 Hierarchical Uppaal

Hierarchical structures are a popular theme in specification formalisms, such as statecharts [Har87] and UML [BRJ98]. The main idea is that locations not necessarily encode atomic points of control, but can serve as an abbreviation for more complex behavior. If a non-atomic location is entered, this may trigger a cascade of events irrelevant to the level of the system that is currently in focus. If a more detailed view is required, the explicit description of the sub-component can be found isolated, since dependencies between the different levels of hierarchy are restricted.

The immediate benefit is a concise description, that allows a complex system to be viewed at different levels of abstraction and nevertheless contains all information in detail. Moreover, symmetries can be expressed explicitly: If two sub-components A and B of a super-state S are structurally identical, they may be described as instantiations of the same template (with possibly different parameters). Copies of states may exist for notational convenience, ambiguities are resolved by a unique-name assumption.

We believe that Uppaal can benefit greatly from these concepts, since they support a cleaner and more structured design of large systems. The model can be constructed top down, starting with a very abstract notion that is refined subsequently. The simulator can then be used to validate that the model coincides with the intuition of the designer. Moreover, it is possible to reason about the model with arbitrary granularity, since, e.g., safety- and deadlock-properties can be model-checked at each stage of modeling. The refinement relation is then given by purely structural information.

A second—however ambitious—goal is to exploit the structure in shaping more efficient model-checking algorithms. Related work [AW99] indicates that locality of information can be exploited straightforward in reachability analysis. Also, the work in [BKLHLN99] indicates that – at least for untimed systems – one may exploit the hierarchical structure of a system during analysis. However, in the setting of Uppaal, this is more difficult, since all parallel processes implicitly synchronize on the passage of *time*. Approaches for local-time semantics [BJLY98] have also yet to be shown to improve verification time in reasonable scenarios, i.e. where the dependency between parallel subcomponents is low, thus not all interleavings have to be taken into account. As a first step towards this, we work on a careful definition of hierarchical timed automata, that support encapsulation and local definitions. In particular, the synchronization of joins raises semantic problems that can be resolved in various ways. Since some of the design choices are not obvious at first, case-studies are planned that corroborate the naturalness of this definition in complex examples. A translation of hierarchical timed automata into a parallel composition of flat ones serves to readily provide prototypes that can corroborate decisions here or detect clumsy choices. This flattened system necessarily contains auxiliary constructs to imitate the behavior of the hierarchical ones. We expect the case-studies to give an intuition, whether this translation slack is tolerable.

The design of the hierarchical timed automata is meant to be close to UML state-chart diagrams. As for the real-time aspect, one output of this considerations will be a real-time profile[1], that suggests an extension of UML formalisms with clocks and timed invariants. This work is carried out in the context of AIT-WOODDES project No IST-1999-10069.

[1]A profile is the standard formal way to extend UML concepts.

References

[AJ01] Tobias Amnell and Pontus Jansson. Report from astec-rt auto project — central locking system case study. In preparation., 2001.

[AW99] Rajeev Alur and Bow-Yaw Wang. "Next" Heuristic for On-the-fly Model Checking. In *Proceedings of the Tenth International Conference on Concurrency Theory (CONCUR'99)*, LNCS 1664, pages 98–113. Springer-Verlag, 1999.

[BFH$^+$] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Larsen, Paul Pettersson, and Judi Romijn. Efficient guiding towards cost-optimality in UPPAAL. To be submitted to TACAS'2001.

[BFH$^+$00] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-Cost Reachability for Priced Timed Automata. Submitted for publication. Available at `http://www.docs.uu.se/docs/rtmv/-papers/bfhlprv-sub00-1.ps.gz`., 2000.

[BHV00] Gerd Behrmann, Thomas Hune, and Frits Vaandrager. Distributing timed model checking – How the search order matters. In E. Allen Emerson and A. Prasad Sistla, editors, *Proc. of the 12th Int. Conf. on Computer Aided Verification*, number 1855 in Lecture Notes in Computer Science, pages 216–231. Springer–Verlag, 2000.

[BJLY98] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial Order Reductions for Timed Systems. In *Proc. of CONCUR '98: Concurrency Theory*, 1998.

[BKLHLN99] G. Behrmann, H. Andersen K. Larsen, H. Hulgaard, and J. Lind-Nielsen. Verification of hierarchical state/event systems using reusability and compositionality. In *Proc. of the 5*th *Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science. Springer–Verlag, 1999.

[BLP$^+$99] Gerd Behrmann, Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Efficient timed reachability analysis using clock difference diagrams. In *Proc. of the 11th Int. Conf. on Computer Aided Verification*, Lecture Notes in Computer Science. Springer–Verlag, 1999.

[BRJ98] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[CL00] Franc Cassez and Kim G. Larsen. The impressive power of stopwatches. In *Proc. of CONCUR '2000: Concurrency Theory*, 2000.

[DY00] Alexandre David and Wang Yi. Modelling and analysis of a commercial field bus protocol. In *Proc. of 12th Euromicro Conference on Real-Time Systems*. IEEE Computer Society Press, June 2000.

[Feh99] Ansgar Fehnker. Scheduling a steel plant with timed automata. In *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications (RTCSA99)*, pages 280–286. IEEE Computer Society, 1999.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 1987.

[HLP00] Thomas Hune, Kim G. Larsen, and Paul Pettersson. Guided Synthesis of Control Programs Using UPPAAL. In Ten H. Lai, editor, *Proc. of the IEEE ICDCS International Workshop on Distributed Systems Verification and Validation*, pages E15–E22. IEEE Computer Society Press, April 2000.

[HRSV] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits Vaandrager. Linear parametric model checking of timed automata. Accpeted for Tools and Algorithms for the Construction and Analysis of Systems, 2001.

[Hun99] Thomas Hune. Modelling a real-time language. In *Proceedings of FMICS*, 1999.

[IKL$^+$00] Torsten K. Iversen, Kåre J. Kristoffersen, Kim G. Larsen, Morten Laursen, Rune G. Madsen, Steffen K. Mortensen, Paul Pettersson, and Chris B. Thomasen. Model-checking real-time control programs — Verifying LEGO mindstorms systems using UPPAAL. In *Proc. of 12*th *Euromicro Conference on Real-Time Systems*, pages 147–155. IEEE Computer Society Press, June 2000.

[KLPW99] K. Kristoffersen, K. Larsen, P. Pettersson, and C. Weise. VHS Case Study 1 - Experimental Batch Plant using UPPAAL. BRICS, University of Aalborg, Denmark, May 1999.

[LBB$^+$01] Kim G. Larsen, Gerd Behrmann, Ed Brinksma, Ansgar Fehnker, Thomas Hune, Paul Pettersson, and Judi Romijn. As Cheap as Possible: Efficient Cost-Optimal Reachability for Priced Timed Automata. Submitted for publication., 2001.

[LNAB$^+$98] Jørn Lind-Nielsen, Henrik Reif Andersen, Gerd Behrmann, Henrik Hulgaard, Kåre J. Kristoffersen, and Kim G. Larsen. Verification of Large State/Event Systems Using Compositionality and Dependency Analysis. In Bernard Steffen, editor, *Proc. of the 4*th *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 201–216. Springer–Verlag, 1998.

[LPY97] 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.

[LWYP99] Kim G. Larsen, Carsten Weise, Wang Yi, and Justin Pearson. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.

[Pet99] Paul Pettersson. *Modelling and Analysis of Real-Time Systems Using Timed Automata: Theory and Practice*. PhD thesis, Department of Computer Systems, Uppsala University, February 1999.

[SD97] U. Stern and D. L. Dill. Parallelizing the Murφ verifier. In Orna Grumberg, editor, *Proc. of the 9th Int. Conf. on Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 256–267. Springer–Verlag, June 1997. Haifa, Isreal, June 22-25.