Tools for Real-Time UML: Formal Verification and Code Synthesis

Tobias Amnell[1], Alexandre David[1], Elena Fersman[1], M. Oliver Möller[2], Paul Petterson[1] and Wang Yi[1]

[1] Department of Information Technology, Uppsala University, {tobiasa,adavid,elenaf,paupet,yi}@docs.uu.se
[2] BRICS[***], Department of Computer Science, Aarhus University, omoeller@brics.dk

Abstract. We present a real-time extension of UML statecharts to enable modelling and verification of real-timed constraints. For clarity, we shall consider a reasonable subset of the rich UML statechart model and extend it with real-time constructs (clocks, timed guards, invariants and real-time tasks). We have developed a a rule-based formal semantics for the obtained formalism, called *hierarchical timed automata* (HTA). To use the existing tool UPPAAL for formal verification, HTA are translated to enriched timed automata model. We report on an XML based implementation of the translation from HTA to a network of timed automata and present an example to illustrate our technique and report run-time data for the formal verification part. We also report on a prototype implementation of the code synthesis for the *legOS* platform.

1 Introduction

In this short paper, we briefly discuss two recent developments that are aimed to adopt the Uppaal tool [LPY97] towards UML statechart compability.

First, we propose a timed version of the UML statecharts model appropriate for high-level design and analysis of real-time systems. The model has a simple and clear semantics, and is powerful enough to describe the characteristic features of real-time systems, such as scheduling algorithms, timers, timeouts, continuous behaviors, etc. We refer to the formalism as *hierarchical timed automata* (HTA). HTA are defined from a subset of UML statecharts extended with clocks, integer variables, and channel synchronizations. In the next section, we present the chosen UML subset and the real-time extensions. We also outline the formal syntax and semantics of HTA.

Secondly, we present a model called *executable timed automata* as an extension of timed automata allowing code synthesis. Most commercial real-time tools provide a code generation capability. However the translation is often done process by process without taking allocated resources such as total CPU utilization into account. For the proposed model of executable timed automata we are able to synthesize code with predictable timing behaviour.

The overall goal is a uniform framework for modeling, analysis, and code synthesis of real-time systems. Figure 1 illustrates the current position of the Uppaal tool in the context of UML. The tools exchange data through a XML/XMI interface mechanism. The UML tools need a translation for compatibility with the specific formalism used. Then simulation, verification, debugging with trace analysis, and ultimately code synthesis are carried out.

2 Hierarchical Timed Automata

Our HTA is essentially a subset of UML statecharts extended with real-time constructs. State are either basic or composed of state machines themselves. Transitions are guarded, can entail synchronization, and update local or global variables. For the real-time aspects, clocks, invariants, and timed guards are added. In the following, we describe in some more detail, the chosen subset of UML statecharts and the real-time extensions.

[***] Basic Research in Computer Science, Centre of the Danish National Research Foundation.

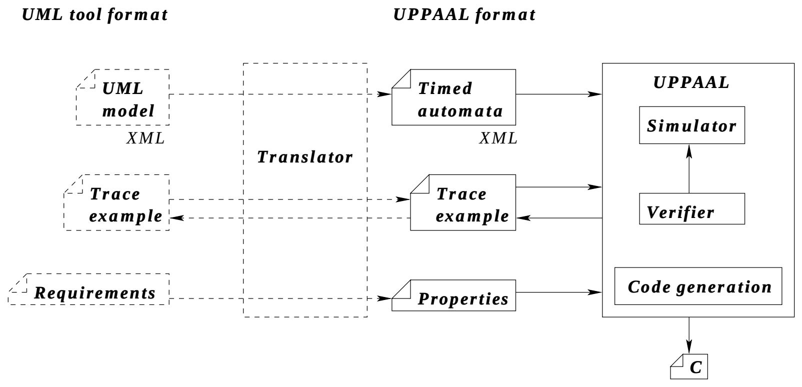

Fig. 1. Modelling, analysis, and code synthesis framework.

Statecharts Subset. The UML statecharts subset is a hierarchically organized state machine with *XOR* states, *AND* states and basic states. *XOR* and *AND* states are the building blocks of the state hierarchy. They contain other *sub states*. In *XOR* states, sub states are related to each other by "exclusive-or", i.e. in an active *XOR* state exactly one sub state is active at a time. In an active *AND* states, all sub states are active, i.e., they run in parallel. *AND* states cannot have basic sub states. Note that the set of active states is a dynamic property of the system.

In HTA we do not allow transitions crossing hierarchy levels as in UML statecharts. Instead we always use explicit entry and exit points represented as UML stubs. This makes it easier to give a clear semantics and perform formal analysis. However, this does not have to hinder designers since a sufficiently advanced editor can provide UML facilities by keeping the explicit representation internally. Furthermore, since every legal UML model has a corresponding representation in the HTA notation, we do not restrict the expressiveness.

HTA include *shallow* history as a special entry. Deep history is not addressed explicitly, since it is straight forward to encode deep history using shallow history. Parallel components communicate via CCS style handshaking synchronisations [Mil89]. The more general event model of UML can be encoded via broadcast communication which will be supported in the near future.

Real-Time Extensions. We equip our HTA model with clocks, invariants, and urgency to capture real-time behavior. *Clocks* are real-valued variables that increase their values synchronously (i.e. at the same rate) whenever time elapses. All clocks start with the initial value 0 and may be reset to 0 along transitions. This extension is influenced strongly by the timed automata model [AD94] that has been studied thoroughly in the literature and has been successfully applied in formal verification.

Invariants are boolean expressions that can be associated to states and are used to enforce progress. They must evaluate to *true* as long as the state is active, i.e. the state must be left before the invariant evaluates to *false*. This notion of progress is expressive enough to imitate the run-to-completion step concept of UML statecharts.

Urgency is a concept that gives priority to actions over time delay. We propose to use this as a property of transitions. If *urgent transition* is enabled, no time delay is possible, i.e., the next step is an action step.

Formal Semantics of HTA. The complete formal syntax and semantics are defined in [DM01]. We give a brief sketch. Semantics is based on the configuration vector (ρ, μ, ν, θ) that is a snapshot of the system. $\rho : S \to 2^S$ captures the control location. It is a partial version of δ. $\mu : S \to (\mathbb{Z})^*$ gives the valuation of the local integers of a state. $\nu : S \to (\mathbb{R}^+)^*$ gives the real valuation of the clocks visible at a given state. θ reflects the history and has a state and a variable component.

The semantics is then defined in term of a configuration vector transformation $\mathcal{T}_t(\rho, \mu, \nu, \theta)$ that performs a join, a transition, and a fork. We give here two of the rules:

$$\frac{\mathit{TransitionEnabled}(t : l \xrightarrow{g,r,u} l',\ \rho, \mu, \nu)}{(\rho, \mu, \nu, \theta) \xrightarrow{t} \mathcal{T}_t(\rho, \mu, \nu, \theta)}\ \mathit{action}$$

$$\frac{\mathit{Inv}(l)(\rho, \nu + d) \qquad \neg\, \mathit{UrgentEnabled}(\rho, \mu, \nu)}{(\rho, \mu, \nu, \theta) \xrightarrow{d} (\rho, \mu, \nu + d, \theta)}\ \mathit{delay}$$

All the work resides in defining the predicates *TransitionEnabled, UrgentEnabled,* and *Inv* that evaluate when a given transition is enabled with respect to a cascade join-transition-fork, when urgent transitions are enabled with respect to synchronizations between transitions, and when invariants are enabled.

Case Study: Pacemaker Example. We have implemented a translator from HTA to UPPAAL timed automata to experiment with formal verification of HTA and to compare the sizes between a HTA and its corresponding representation as timed automata. The modeled system is the well-known pacemaker example (see [DM01] for details). Using the implemented translation, we have been able to check a number of safety and liveness properties of the system, indicating that formal verification of HTA is indeed feasible. Table 1 shows a comparison between the input and the output of the translator.

	HTA model	UPPAAL model
# control locations	35	45
# transitions	47	177
# variables and constants	33	72
# clocks	6	6

Table 1. Translations of a hierarchical timed automaton description to an equivalent flat UPPAAL model.

3 Code Synthesis

We propose a way to synthesize code from timed automata models that will have predictable timing behavior. Inspired by the design philosophy of synchronous languages e.g. Esterel [BG92], we assume that the underlying real-time operating system guarantees the *synchrony hypothesis*[1]. We extend timed automata so that each node of an automaton is associated with 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 is shown in Figure 2.

Intuitively, a discrete transition in an extended 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 2, 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.

Schedulability analysis. In the extended model, the tasks in the ready queue are executed according to a chosen scheduling strategy, e.g. earliest deadline first. In [EWY99] it is shown that the schedulability of extended automata can be checked by reachability analysis for non-preemptive tasks. It is equivalent to prove that all schedulable states are schedulable. We are working on extending this result to more general execution models.

[1] That is, the underlying operating system calls take little time compared to the worst case execution times and deadlines of tasks.

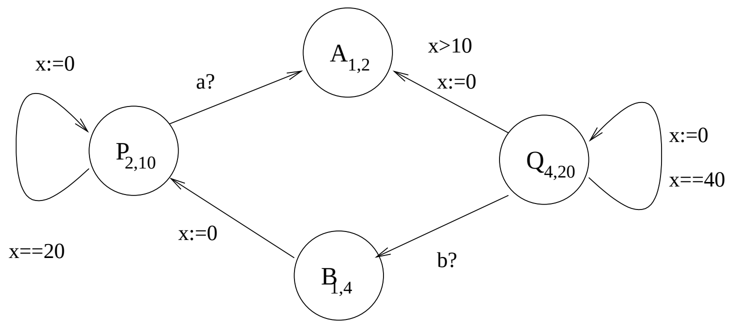

Fig. 2. The system shown consists of 4 tasks as annotation on nodes, where P, Q are periodic with periods 20 and 40 respectively (specified by the constraints: x==20 and x==40), and A, B are sporadic or event driven (by event a and b respectively). The pairs in the nodes give the computation times and deadlines for tasks e.g. for P they are 2 and 10 respectively.

Synthesizing Code for Controllers. Timed automata annotated with tasks, as described above, are used as a design model for control programs. The discrete transitions (i.e. the control structure) of the automaton and the associated task are implemented using a small set of (common) system calls and light-weight threads provided by the underlying real-time operating system. This allows us to synthesize code for a variety of 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 which includes an 8-bit Hitachi micro-controller. We belive that this hardware is rather typical for embedded systems of the kind we are targeting. Therefore our prototype give some promising evidence that our approach is viable.

4 Conclusion

We are approaching a situation where the Uppaal tool can manage imported UML statechart models represented as HTA. Code generation from Uppaal models by means of higher level concepts is work in progress. This opens a way to perform code generation and formal verification in the same framework. Preliminary experiments are encouraging.

References

[AD94] Rajeev Alur and David Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 2(126):183–236, 1994.

[BG92] G. Berry and G. Gonthier. The Synchronous Programming Language ESTEREL: Design, Semantics, Implementation. *Science of Computer Programming*, 19:87–152, 1992.

[DM01] Alexandre David and M. Oliver Möller. From HUppaal to Uppaal: A translation from hierarchical timed automata to flat timed automata. Research Series RS-01-11, BRICS, Department of Computer Science, University of Aarhus, March 2001.

[EWY99] Christer Ericsson, Anders Wall, and Wang Yi. Timed Automata as Task Models for Event-Driven Systems. *Proceedings of RTSCA'99*, 1999.

[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.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice–Hall, 1989.