Extending HyCharts with State-Invariants

Thomas Stauner*

Institut für Informatik, Technische Universität München
D-80290 München, Germany
http://www4.in.tum.de/
Email: stauner@in.tum.de

Abstract. The paper defines an extension of the HyChart notation [8] with state-invariants. HyCharts are a graphical, modular, and formal description technique for the specification of hybrid, i.e. mixed discrete-continuous, systems. Methodological issues arising from the extension are discussed.

1 Introduction

HyCharts consist of two modular, visual description techniques for the specification of hybrid, i.e. mixed discrete and continuous, systems. *HyACharts* are used for the specification of the system architecture, and *HySCharts* specify the components' behavior. HySCharts can be seen as a hybrid extension of the Statechart variant ROOMcharts [12]. The basic step of this extension is to annotate every state with an *activity* which specifies how continuous variables evolve when control is in the respective state. [8] introduces HyCharts, and [6] presents the underlying theory in more detail.

In contrast to the popular hybrid automata formalism, transitions in HySCharts are *eager*, i.e. they must be taken as soon as they are enabled. Transitions in hybrid automata [1] are *delayable*. They can, but need not, be taken as long as their guard is true. In order to be able to express that a transition has to be taken under certain circumstances, hybrid automata use state invariants. A state must be left before its invariant becomes false.

Pros and cons of invariants. Various case studies [14,7,11] have demonstrated that the eager transitions of HyCharts are useful in practice and eliminate some problems that occur in the context of delayable transitions and state invariants (see Section 2). Nevertheless, delayable transitions are useful in some applications.

First, they are useful whenever it is necessary to express a lack of knowledge of exact timing in a system. Often such a lack of knowledge exists when modeling

* This work was supported with funds of the Deutsche Forschungsgemeinschaft under reference number Br 887/9 within the priority program *Design and design methodology of embedded systems*.

the environment of embedded systems. Incoming messages in a communication network can, for example, be conveniently modeled with delayable transitions. In [14], for instance, nondeterministically drifting clocks were used to express the varying delay between incoming messages, whereas using delayable transitions instead such as in [3] might have been more natural. While delayable transitions can therefore be helpful in modeling the environment of an embedded system, eager transitions are what is usually needed when modeling the software part of the embedded system. In fact, a considerable amount of effort is spent in [3] to make transitions of the software part of the considered system eager.

Nevertheless, there is a further aspect which indicates that delayable transitions are also needed for the software part of an embedded system. In our view notations such as hybrid automata can beneficially be used for requirements capture and the early design steps of hybrid, embedded systems [10]. In theses phases designers usually want to express that some actions are taken when certain conditions are satisfied. They are not so much interested in the detailed timing, i.e. in possible small delays between the satisfaction of conditions and the execution of the corresponding actions. They are, however, aware that such delays exists when the model is implemented on digital computers. One primary reason for such delays is that the software can only sense the status of its environment within the discrete time grid given by the hardware. While we think it is not adequate to already think about sampling rates and timing uncertainties in detail in early design phases, models must nevertheless be designed in a way that tolerates small delays without violating vital system properties. Otherwise, if the model relies on the absence of any delays, it cannot be implemented later on. Thus the models of the system specified in the requirements capture phase and the early design phase should allow timing uncertainties. In order not to bother the designer with these uncertainties in the beginning we propose tool support which automatically computes state invariants that allow timing uncertainties whenever the designer enters new transitions. For instance, for a transition guard $x \geq c$ the tool would generate invariant $x < c + \epsilon$ resulting from the negation of the guard plus some overlap with the guard, $\epsilon > 0$.

A prerequisite for such a methodology is a notation which allows to model delayable transitions in a convenient way. While delayable transitions can already be expressed indirectly in the current version of HySCharts by using nondeterministic activities, this paper presents a semantic extension of HySCharts by invariants which allows modeling delayable transitions in a more straightforward, explicit way.

In further design steps the timing uncertainty can be used by the designers to select a sampling rate for the system that guarantees that transitions are taken within the delay permitted by the model. Hence, notations with delayable transitions are useful in this context because they allow to regard the step to a model with a fixed sampling rate as a refinement step. Note that sampling rate selection is constrained by the dynamics of the underlying physical environment and the functionality required from the embedded system. It is not clear from

the beginning, except if the dynamics is already well known from legacy systems, whose only part to be updated is the software.

Overview. The rest of the paper is organized as follows. Section 2 discusses a number of problems that can occur in the context of state invariants. It also indicates related work. In Section 3 HySCharts are extended with state invariants. First, we elaborate how HySCharts are annotated with invariants and how the state transition relation is derived. Then, the well-definedness of the resulting components is proven. Finally, we draw some conclusions.

2 Problems with Invariants

The most popular formalism for modeling hybrid and real time systems which allow delayable transitions are hybrid automata [1]. Unfortunately the combination of transitions guards and invariants can easily lead to time deadlocks in hybrid automata and also in some timed automata dialects, as e.g. the one supported by the UPPAAL model checker [9]. A time deadlock results when the invariant of the automaton's current state does not allow any further time progress while none of the transitions emerging from the state is enabled. In such cases no further actions, neither time progress nor executing a transition, is possible. As time deadlocks cannot occur in real systems they are considered specification errors. In practice it is rather difficult to detect such errors [13].

Bornot et. al. therefore propose a formalism that avoids time deadlocks by eliminating invariants and using *deadlines* for the transitions instead [2]. In the proposed formalism, *timed automata with deadlines*, a transition may be taken as soon as its guard is true. It must be taken as soon as its deadline becomes true. The combination of deadlines and guards allows to define various notions of urgency of transitions. The authors introduce a number of such notions and argue that they are useful for modeling multimedia systems. [2] furthermore notes that specifying eager transitions is sometimes difficult in formalisms based on invariants. Namely, hybrid automata require that the invariant of a state is true when the state is entered. In this context it is often difficult to specify an eager transition which is already enabled when the state is entered. To express such a case the invariant must be chosen in a way which ensures that it is true when the state is entered, but only remains true for this very moment. In the HySChart extension we propose in the following a state may be entered with its invariant being false. However, no time may elapse in a state whose invariant is false. The above problem is therefore alleviated in HySCharts.

The so called phase transition systems given in [4] allow to model delayable transitions by providing a minimum and maximum delay for each transition. A transition can be taken if it has been continuously enabled for its minimum delay and it must be taken if it has been continuously enabled for its maximum delay. While this concept is adequate for real-time systems and for discrete, message based communication, we think that for hybrid systems, the permitted

minimum and maximum delay of transitions is a derived concept, resulting from the desired properties of the system under development and the dynamics of the underlying physical system. For instance, one desired property often is that some continuous variable always remains within given bounds. The bounds together with the variable's possible rate of change implies the permitted maximum delay with which a system has to react when the variable's value comes close to the border of the given bounds. In our opinion it is therefore more natural to think about permitted deviations in some variables' values first ("how close to the border should the variable get"), before fixing permitted time delays. A way of expressing these deviations is via invariants which overlap with the transition guards.

3 Invariants for HyCharts

There are two basic possibilities for the extension of HyCharts by invariants. First, invariants could be introduced as a syntactic macro which mimics the nondeterminism w.r.t. the time when a transition is taken by a nondeterministic assignment and a nondeterministic activity for a new auxiliary variable.

The second possibility, which we will pursue, introduces invariants as a new semantic concept. We pursue this alternative as it does not require blowing up a model with auxiliary variables. Furthermore the semantic extension is very similar to the HyChart definition used so far. Note that the old HyChart definition which only allows eager transitions is incorporated in the new version as a special case.

3.1 Extension of the State Transition Relation

The visual notation of HySCharts corresponds to that of ROOMcharts [12], but extends primitive and hierarchic states with a further label (written in Italics) which refers to an *activity*, i.e., to a predicate which specifies how continuous variables evolve when control is in the respective state. The semantics definition of HySCharts is based on the perception that the visual representation of a hierarchic state machine may be regarded as a hierarchic graph being built of from nodes and arcs and operators on these nodes and arcs, such as sequential composition or branching.[1] As HySCharts describe two aspects of a system, namely a pure state transition relation without time and the continuous evolution of the variables over time resulting from the activities, [6] defines two transformations of a HySChart into two hierarchic graph dialects. The first transformation extracts the state-transition information (or the *combinational part Com*), the second one extracts the information describing continuous evolution (or the *analog part*

[1] Note that we use the term *node* when we talk about the graphical representation of a state in a HySChart.

Ana). For the extension of HySCharts with invariants only the first transformation needs to be modified, which will be considered in the following. A tuple $(i, k.s, k'.s')$ in the state transition relation Com consists of the current external input values i, the present state $k.s$ and the next state $k'.s'$. A state $k.s$ is built up from of the *control state* k corresponding to a node in the HySChart and the *data state* s which is a valuation functions for the HySChart's variables.

From HySCharts to hierarchic graphs. Syntactically we extend HySCharts by labeling (basic and hierarchic) nodes with invariant names, written in normal font (see the node in Fig. 1, left). The invariant labels refer to predicates constraining the values of the HySChart's variables. In the transformation from a HySChart to a hierarchic graph which specifies the state transition relation encoded in the diagram, the primitive and hierarchic nodes in a HySChart are regarded as macros for *computation units*. A computation unit gets the *control* (or *state*) along one of its *entry points* $\texttt{en}_i$ and returns the control (or *next state*) along one of its *exit points* $\texttt{ex}_j$ (Fig. 1, top right). After getting control along a regular entry point, i.e., an entry point different from *wait* `wt`, a computation unit first executes its *entry action* `entry`. Then it evaluates a set of *action guards*.[2] If one of the guards is true, then the corresponding *action* is said to be *enabled* and its *body* is executed. After finishing its execution, the computation unit executes its *exit action* `exit`. Finally, control is given to another computation unit along the exit point corresponding to the executed action.

If more than one guard is true, then the computation unit *nondeterministically* chooses one of them. If none of the guards is true, then the discrete computation is completed, and the control leaves the computation unit along the designated wait exit point `wt`. The guard associated with the wait exit point is called *wait exit action*. In the current version of HySCharts it is defined as the negation of the disjunction of the guards of the actions $\texttt{action}_k$, emerging directly from the regarded node (and from none of its subnodes). The hierarchic graph containing the activity information *Ana* takes advantage of the information about the exit point to determine the activity to be executed and gives control back along the corresponding wait entry point of the graph for the state transition relation Com. Entering and leaving a computation unit along its wait entry and wait exit point may be regarded as waiting or *idling* in the corresponding state until a transition is enabled. The state transition relation results from the composition of such computation units. Due to the composition, a next control state k' given by Com encodes the wait exit point on which some computation unit is left (and similar for present control state k and wait entry points).

In the reduction from HySChart *with* invariants to additively interpreted hierarchic graphs, the idea is to transform invariants to wait exit actions. This means that instead of using the negation of the disjunction of the action guards of a node's outgoing transitions as wait exit action, we now use the invariant specified for that node as its wait exit action. The transformation of a primitive or a hierarchic node with invariant (Fig. 1, left) to a hierarchic graph is

[2] An action $\texttt{action}_k$ consists of a *guard* and a *body*.

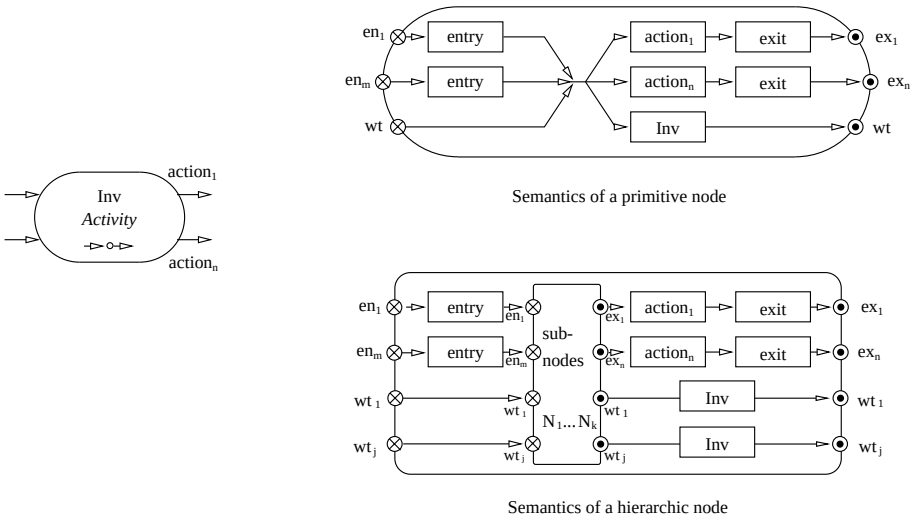

Fig. 1. Semantics of HySCharts with invariants.

depicted in Fig. 1, top right, for primitive nodes and Fig. 1, bottom right, for hierarchic nodes, where `Inv` refers to the invariant. The reduction for a node with preemptive transitions, i.e. with transitions which directly emerge from a hierarchic node and from node of its subnodes, is similar. Starting from the hierarchic node in Fig. 2, left, first the semantics of its composed subnodes is computed and a diagram like Fig. 2, top right, is obtained. It is then converted to Fig. 2, bottom, which incorporates the preemptive actions $\mathtt{pa}_1$ to $\mathtt{pa}_h$ in the resluting hierarchic graph. To be consistent with wait exit actions, the invariant labels refer to relations between the current input, the present data-state and the next data-state, $Inv \subseteq \mathcal{I} \times \mathcal{S} \times \mathcal{S}$.[3] Unlike other actions, invariants may not change values. Hence, we require that the next state is equal to the present state, formally $(i, s, s') \in Inv \Rightarrow s' = s$. By convention, all nodes which are not labelled with invariant labels have the negation of the disjunction of the action guards of the transitions emerging directly from the node as invariant. Due to this convention nodes without invariant label have eager transitions, which is compatible with the old definition of HySCharts in [6].

Sound invariants. In order to avoid time deadlocks we demand that for every node there always either is a transition emerging directly from it (and from none of its subnodes) which is enabled, or the invariant holds. In case of a hierarchic node its invariant in the derived hierarchic graph can only be reached when coming from the wait exit points, and hence from the invariants, of its subnodes (see the arcs leading to boxes labeled `Inv` in Fig. 1, bottom right, and Fig. 2, bottom). We therefore demand that if the invariant of one of its subnodes is true, either the hierarchic node's invariant also is true, or one of the transitions directly emerging from it is enabled. Formally, to avoid time deadlocks it suffices

[3] Note that such a relation is usually given by a predicate.

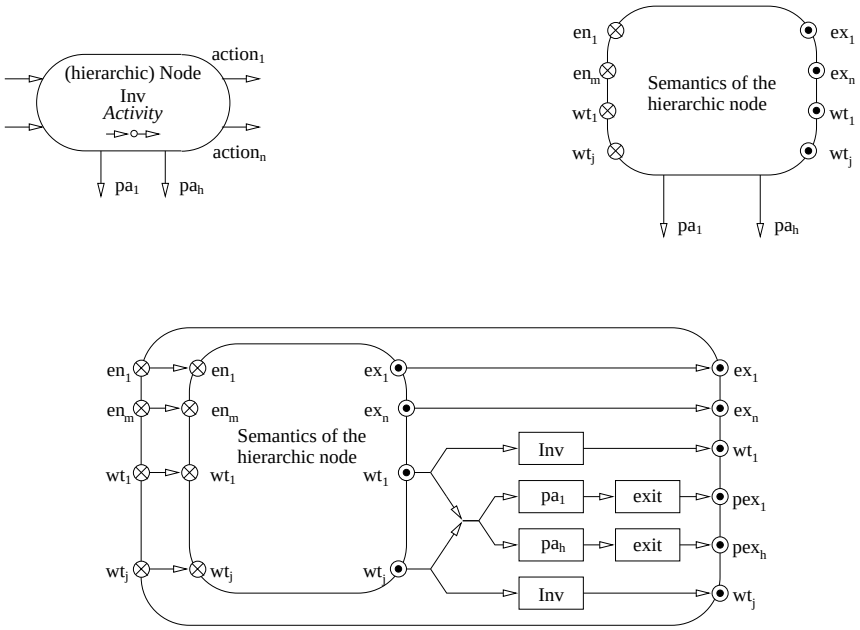

Fig. 2. The semantics of preemption.

to require that there exists $(i, s', s'') \in Inv \cup \bigcup_{i=1}^{n} Guard_i$ for all $(i, s, s') \in \bigcup_{j=1}^{k} Inv_j$, where the $Guard_i$ refer to the action guards of transitions directly emerging from the considered node and the Inv_j refer to the invariants of its subnodes. For every node we may only consider those transitions which emerge directly from it (i.e. preemptive transitions), because exit from the node on the other transitions is determined by the actions associated with the subnodes. Hence, if the invariant of a subnode is true, either the invariant of its enclosing hierarchic node must also be true or one of its preemptive transition must be enabled. For a hierarchic node without preemptive transitions this means that its invariant must be implied by the disjunction of the invariants of its subnodes.

For semantic reasons we furthermore require that the projection of every invariant on the input and present state results in a topologically open set on $\mathcal{I} \times \mathcal{S}$.[4] In the next section we will see that this ensures that if the HySChart's combinational part Com passes control on a wait exit point (i.e. the invariants of the respective hierarchic node are true) some time $\delta > 0$ can progress. Informally arguing on the real numbers, the reason for this is that any point in an open set has a δ-neighborhood which also is in the set.

Note that in contrast to hybrid automata it is not necessary that the invariant of the source or destination state of a transition is true when the transition is

[4] As topology we use the Tychonoff topology on $\mathcal{I} \times \mathcal{S}$ which is induced by using the discrete topologies on the variable domains different from $\mathbb{R}$ and the Euclidean topology on $\mathbb{R}$ for the variable domains that are equal to $\mathbb{R}$ [5].

taken. The hierarchic graph for a primitive node explains this. The invariant is not checked when the node is entered or left on an exit point different from wait. It is only checked, when the node is left on its wait exit point.

3.2 Well-Definedness of Components with Delayable Transitions

In the preceding section we saw how invariants are reduced to wait exit actions. Now we outline the prove that a component defined by a HySChart with invariants and delayable transitions is well defined.

In the context of HyCharts with invariants we say the combinational part Com *can idle* for state $s \in \mathcal{S}$ and input $i \in \mathcal{I}$ iff s is in the set of possible next states determined by Com for s and i, i.e. iff $s \in Com(i, s)$. Thus, in comparison to the definition of idle for HyCharts without invariants, we do not demand that s is the only possible output of Com here. If for present state s and input i, next state $s \in Com(i, s)$ is indeed selected in the execution thread under consideration, we say that Com *idles* or *is idle* at that point in time.

The semantics of a HySChart basically is defined as the composition with feedback of the (time extended) state transition relation Com with the analog part Ana, together with an infinitely small delay on the feedback. According to its definition in [6] the semantics is a relation consisting of tuples of hybrid input streams and hybrid output streams. A *hybrid stream* is a function from the non-negative real numbers to some domain, $\mathbb{R}_+ \to M$, which is piecewise smooth and piecewise Lipschitz continuous. We say that a function $f \in \mathbb{R}_+ \to M$ is *piecewise smooth and piecewise Lipschitz continuous* iff every finite interval on the non-negative real line $\mathbb{R}_+$ can be partitioned into *finitely* many left closed and right open intervals such that on each interval f is infinitely differentiable and Lipschitz continuous for $M = \mathbb{R}$ or f is constant for $M \neq \mathbb{R}$. As the feedback composition of Com and Ana does not contain a delay $\delta > 0$, the existence of a fixed point is not guaranteed a priori. Instead, it is a consequence of the properties of Com and Ana.

Existence of a fixed point. We substantiate that at any point in time t_0 there exists a $\delta > 0$ such that time δ can pass between two discrete moves by the combinational part or the environment. Above, we demanded that each invariant identifies a topologically open set. The composition of invariants, which reflects the hierarchy in the HySChart, corresponds to the intersection of the open sets on which the invariants are true. As there are only finitely many hierarchy levels in a HySChart, we get that the composed invariant also identifies an open set $E \subseteq \mathcal{I} \times \mathcal{S}$. As the input stream ι is piecewise infinitely smooth, there must be a time $t_1 > t_0$, such that it evolves continuously from now up to t_1. Now suppose s' is an output of the next state relation Com for the current input i and the current state s at time t_0 (see Fig. 3). Hence the invariant holds for s' and Com can idle for the new state s', i.e. $s' \in Com(i, s')$. Due to the properties of the analog part Ana as formalized in [6] the activity it chooses for the start state s' results in an output which also is continuous up to t_1. Continuity of the

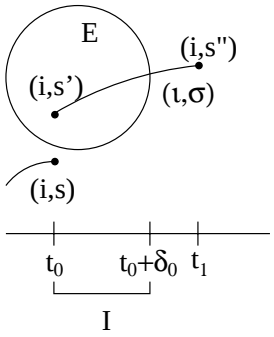

Fig. 3. Computing a permitted delay.

external input and the output of Ana provides that the inverse image I of E is open w.r.t. $[t_0, t_1)$. As $t_0 \in I$, there exists a neighborhood $[t_0, t_0 + \delta)$ of t_0 which is contained in I. Consequently, Com may remain idle for the input and the chosen activity up to time $t' = t_0 + \delta$. Due to the properties of the feedback composition of Com and Ana, the chosen activity is a valid fixed point for it on the interval $[t_0, t')$.

Applying this argument inductively we get a fixed point of the feedback composition of Com and Ana on the interval $[0, \Sigma_{n=0}^{\infty} \delta_n)$ for every initial state s_0. If $\Sigma_{n=0}^{\infty} \delta_n$ diverges, we have a proper fixed point. Otherwise we have a *zeno* execution, the combinational part performs infinitely many discrete moves within a finite interval. Hence, it is not realizable. A sufficient condition for realizability is that there is a lower bound δ on the δ_i for all inputs and initial states. It is in the responsibility of the designer to rule out that his system only permits zeno executions. On the level of semantics, zeno executions are ruled out by the type of HySCharts' semantics which only permits piecewise smooth functions as inputs and outputs.

Note that the restriction to piecewise smooth inputs and outputs is essential for HySCharts with invariants, because the combinational part need not remain idle after it has taken a transition, but can deliberately take a next enabled one. This may cause non-zeno executions which are eliminated by restricting HySChart's semantics to piecewise smooth output.

Moreover, note that open invariants are only a sufficient condition for the existence of a fixed point. What actually is required is that Com may always remain idle for some time until the next transition has to be taken.

4 Conclusion

We defined an extension of the HyChart notation with state-invariants, thereby allowing the convenient specification of delayable transitions, i.e. of transitions which need not be taken at once when they become enabled. We have substantiated the necessity of delayable transitions from a methodological point of view

and proposed to automatically include small delays by the modeling tool. Main motivation for delayable transitions is that demanding that enabled transitions are taken immediately cannot be implemented in software which can only operate with the speed of its underlying hardware. Hence, delays must be taken into consideration at some point of the development process of software for hybrid systems.

Acknowledgment. We thank Alexander Pretschner for his constructive criticism after reading a draft version of this paper.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
2. S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *Proc. of COMPOS'97*, LNCS 1536. Springer-Verlag, 1998.
3. H. Bowman, G. Faconti, J. P. Katoen, D. Latella, and M. Massink. Using UPPAAL for the specification and verification of a lip-sync protocol. Technical Report ERCIM-07/98-R054, CNR - Istituto CNUCE, Pisa, Italy, July 1998.
4. L. de Alfaro and Z. Manna. Verification in continuous time by discrete reasoning. In *Proc. of 4th International Conference on Algebraic Methodology and Software Technology (AMAST)*, LNCS 936. Springer-Verlag, 1995.
5. Ryszard Engelking. *General Topology*, volume 6 of *Sigma Series in Pure Mathematics*. Heldermann Verlag, Berlin, 1989.
6. R. Grosu and T. Stauner. Modular and visual specification of hybrid systems – an introduction to HyCharts. Technical Report TUM-I9801, Technische Universität München, 1998.
7. R. Grosu and T. Stauner. Visual description of hybrid systems. In *Workshop On Real Time Programming (WRTP'98)*. Elsevier Science Ltd., 1998.
8. R. Grosu, T. Stauner, and M. Broy. A modular visual model for hybrid systems. In *Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, volume 1486 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.
9. K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1(1+2), 1997.
10. A. Pretschner, O. Slotosch, and T. Stauner. Developing correct safety critical, hybrid, embedded systems. In *Proc. of New Information Processing Techniques for Military Systems (to appear)*. NATO Research and Technology Organization, October 2000.
11. M. Scherer and A. Trogmann. HyCharts – Hybride Modellierung der ACC. Seminar Work, Technische Universität München, 1999.
12. B. Selic, G. Gullekson, and P. T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons Ltd, Chichester, 1994.
13. T. Stauner. Specification and verification of an electronic height control system using hybrid automata. Diploma thesis, Technische Universität München, 1997.
14. T. Stauner. Specification of (parts of) a lip-sync protocol using HyCharts. In *FBT'99, 9. GI/ITG-Fachgespräch "Formale Beschreibungstechniken für verteilte Systeme"*. Herbert Utz Verlag Wissenschaft, 1999.