Specification of (parts of) a Lip-Sync Protocol Using HyCharts*

Thomas Stauner

Institut für Informatik, Technische Universität München

D-80290 München, Germany

Email: stauner@in.tum.de, http://www4.in.tum.de/~stauner/

Abstract

This paper presents the specification of some parts of a lip-synchronization protocol using HyCharts [GSB98]. The specification is based on a timed automata model for the protocol presented in [BFK+98]. The primary motivation of this work is to examine the utility of HyCharts for problems in the multimedia field, in particular in comparison to timed automata [AD94].

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 [SGW94]. 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. [GSB98] introduces HyCharts and [GS98] presents the underlying theory in more detail.

The purpose of the lip-synchronization protocol, which is regarded in this paper, is to achieve the *acceptably synchronized* presentation of a sound and a video stream, which are sent separately over a network to a presentation device. The protocol is a standard example from the multimedia domain and has been investigated with various formalisms [SHH92, Reg93, ABSS96, BBBC98]. In detail the requirements are as follows [BFK+98]:

- A sound frame must be presented every 30 milliseconds (ms).
- Video frames should be presented 35 to 45 ms after one another. Furthermore, video frames may lag sound by no more than 150 ms and may precede sound by no more than 15 ms.

The protocol assumes that a sound frame arrives every 30 ms. Therefore, timely presentation of sound frames is an easy task. However, no assumption is made about the arrival

*This work was supported with funds of the Deutsche Forschungsgemeinschaft under reference number Br 887/9-1 within the priority program *Design and design methodology of embedded systems.*

rate of video frames. Correspondingly, synchronization cannot be maintained for strongly disturbed video streams.

We do not give the complete specification of the lip-synchronization protocol here. Instead we choose to only have a detailed look at those parts of the specification which are most useful for figuring out differences, advantages and disadvantages of HyCharts in comparison to timed automata. Note that we take a specification based point of view in this paper. Verification is not considered, because there is no verification support available for HyCharts currently.

The parts of the protocol we specify are:

- The overall architecture of the system. In contrast to timed automata, HyCharts include a description technique for the specification of the architecture of a system.

- A model for the sound and video streams. Along the model we demonstrate how non-deterministic activities are employed to model media streams with *jitter*, i.e. with deviations of the timing within the stream. In timed automata non-urgent transitions are used to achieve a similar effect. We will compare these non-urgent transitions with non-deterministic activities from a specification based point of view.

- The video watchdog component, responsible for monitoring the correct timing within the video stream. Along this component we will explain how timeouts are modeled with HyCharts and compare the model with the timed automata model of timeouts.

Sections 2, 3 and 4 introduce and discuss the specification of these parts. In Section 5 HyCharts are compared to timed automata on basis of the lip-sync protocol and the results are summarized. Note that the paper assumes some familiarity with timed automata [AD94]. Furthermore, it does not contain a complete introduction to HyCharts, but explains key concepts informally when needed. For details the reader is referred to [GS98].

2 Architecture of the Lip-Sync Protocol

Figure 1 shows the architecture of the lip-sync protocol as a HyAChart. The video actor *aVideoR* tries to maintain correct timing of the video stream and lip-synchronization with the sound stream. The component *aInitSyncR* takes care of the correct initialization of the protocol. It determines whether sound or video starts first and tells this to the responsible components. The sound actor *aAudioR* tries to maintain correct timing within the sound stream and provides the video actor with a clock value *s_time* telling it the current time within the sound stream. The video and sound streams are modeled within the video and the sound actor, respectively.

As the architecture of the video and the sound actor is largely the same, we will only consider the video actor *aVideoR* in the following. Its architecture is depicted in Figure 2.[1] It consists of a component *aXStream* modeling the incoming video stream. This component tells the video manager *aXManagerR* whenever a new frame has arrived and is ready

[1] Boxes with triangles pointing to the left denote input channels, boxes with triangles pointing to the right denote output channels.

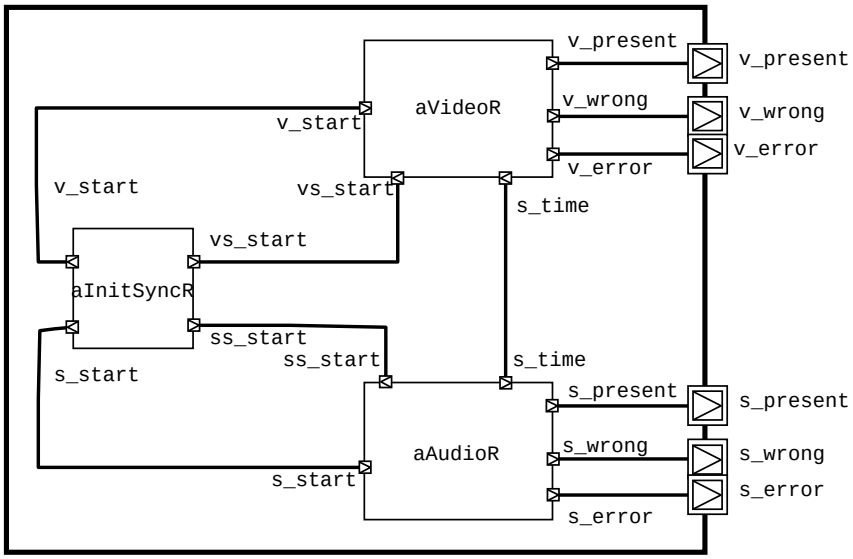

Figure 1: Architecture of the lip-sync protocol.

for presentation. The manager forwards this information to the video sync component *aVSyncR*. The video sync component keeps track of the current time within the video stream. It compares this time with the time within the sound stream *s_time* and decides whether the presentation of a video frame needs to be delayed in order to maintain lip synchronization with it. When the correct time for presentation of a frame is reached it sends a signal *v_ok* to the video watchdog *VWatchdogR*. If no video frame arrives on time an error signal *v_error* is generated.

The video watchdog component measures the time between two succeeding OK signals *v_ok* from the video sync component. As long as each OK signal arrives within the required time distance to its predecessor, the watchdog issues an OK signal *Xokk* to the video manager via the delay component *aDelayR* upon each OK signal it receives. If the time between succeeding OK signals is too small or becomes too great for a "smooth" presentation of the video stream it issues an error signal *v_wrong*.

When the video manager receives the OK signal *Xokk* from the video watchdog it causes the presentation of the next unpresented video frame via signal *v_present*.

The delay component *aDelayR* models the communication delays between the components. It is necessary for mathematical reasons and guarantees causality of the video actor *aVideoR*.

Discussion. Visualization of the architecture greatly facilitates understanding the protocol. First, it depicts the basic components of the system under development. Second, it tells us over which communication channels these components interact. If meaningful names are used for the components and the channels we can get a first idea on how the components work together. In contrast to this, the plain presentation of the individual automata for all the components, as in [BFK+98], does not give much insight at first. In particular it remains unclear which automata communicate with each other. Thus, at lot of explanatory text was needed for the timed automata model in [BFK+98].

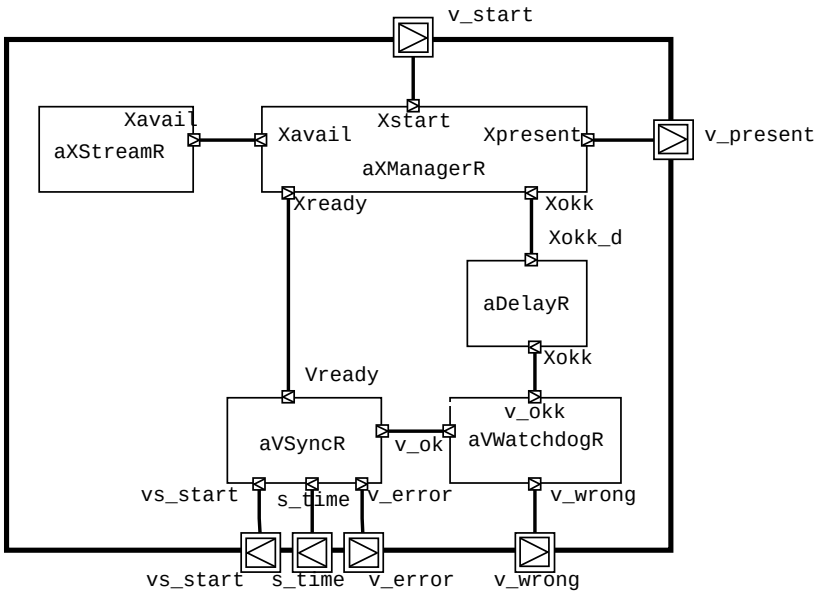

Figure 2: Architecture of the video actor.

3 Models of Media Streams

In both the video actor *aVideoR* and the sound actor *aAudioR* we have a model of the incoming media stream, an actor *aXStream*. The actor *aXStream* simply sends signals *Xavail* to the stream manager, thereby signaling the availability of a new media frame.

In the case of an ideal incoming stream, *aXStream* sends *Xavail* signals precisely every 30 ms for audio frames or every 40 ms for video frames. The point in time were the first frame is sent is chosen non-deterministically. As the model of ideal streams is fairly simple, we omit it here and continue with disturbed media streams.

3.1 Media Streams with Jitter

In this section we regard the model for a media stream that starts at an arbitrary point in time and then continuous to send frames every $p - a$ to $p + b$ time units, where p is the ideal time of playout and a and b denote disturbances from the ideal timing.[2] We can interpret these disturbances as being caused by a source of the media stream which wants to send a frame every p time units, but only has a drifting clock to measure time.

The HySChart for a media stream with jitter is given in Figure 3. The labels refer to the following activities and actions:

$$x\_arb \equiv \dot{x} \in (0, \infty) \qquad x\_drift \equiv \dot{x} \in [c, d]$$
$$init \equiv send \equiv x = p \ \wedge \ Xavail! \qquad entry_{play} \equiv x' = 0$$

The activity x_arb models that x increases at an arbitrary non-negative rate. Together with the action *init* this ensures that the first frame can be sent at an arbitrary point in

[2] According to the terminology from [BFK+98] this is called non-anchored jitter.

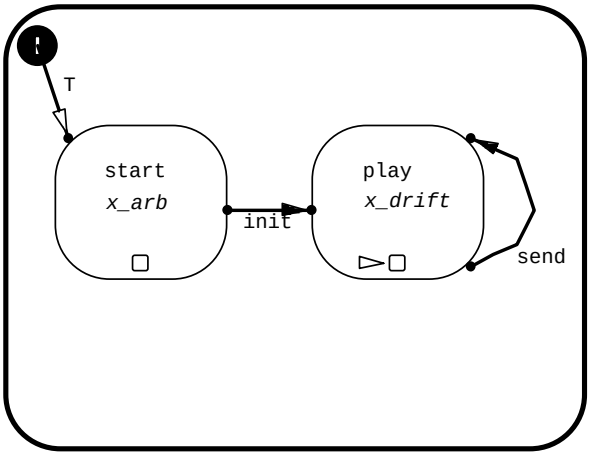

Figure 3: Behavior of a media stream with jitter.

time after the start of the system.[3] Activity x_drift models that x increases at a rate in $[c, d]$ in state *play*. Thus, x can be seen as a drifting clock. The constants are defined as $c = \frac{p}{p+b}$ and $d = \frac{c}{p-a}$ such that a *send* action is performed every $p - a$ to $p + b$ time units. The entry action of *play* resets x. The initial state is *start* with $x = 0$.

Discussion. In contrast to the timed automata model for media streams with (non-anchored) jitter the non-determinism with respect to the time when an action is performed is replaced by non-deterministic activities in the HySChart. From a more abstract point of view this can be seen as replacing a component that does not quite know when to perform an action by one that does know the ideal time instant, but only has a drifting clock.

Note that in contrast to the timed automata model of media streams with jitter [BFK+98] no frame can be transmitted at time $t = 0$. To enable this, a further state would have to be added to our specification.

Timed automata use synchronous communication, the sender of an event is blocked if the receiver is not ready to receive it. In the presence of time dependent state invariants this can easily stop time progress. Therefore, the authors of the timed automata model for media streams had to assume that output of a frame is always possible, i.e. not restricted by the environment. Due to the asynchronous communication in HyCharts – the sender cannot be blocked, an event is simply lost if the receiver is not listening – we need no such assumption.

[BFK+98] presents a further model for another kind of jitter which we omit here.

4 The Video Watchdog Component

From the point of view of comparing a HyChart model of the lip-sync protocol with the timed automata model, the video watchdog component *VWatchdogR* is interesting as it

[3] Like in the timed automata model, it is possible that no frame is sent at all. This can happen if $\dot{x}$ converges to zero. Note that we could enforce that a frame is sent eventually by defining $x\_arb \equiv \dot{x} \in (0, \infty) \ \wedge \ \ddot{x} = 0$.

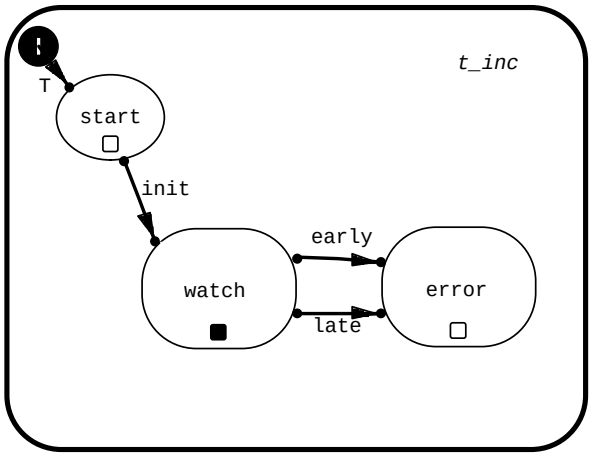

Figure 4: Behavior of the video watchdog component (top level).

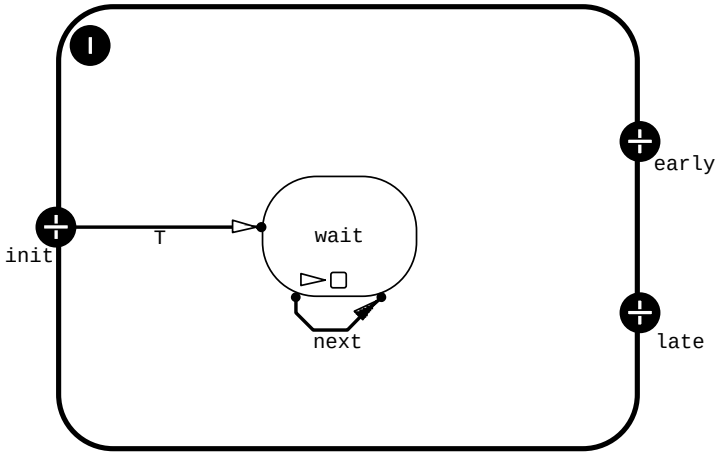

Figure 5: Substate *watch* of the video watchdog.

can be used to demonstrate how to model the important concept of timeouts.

As indicated in Section 2, the purpose of the watchdog component is to monitor whether the video frames, which already were synchronized with the audio frames by the *aVSyncR* component, can be presented at the required rate of one frame every 35 to 45 ms.

The HySChart for *VWatchdogR* is given in Figure 4. The state *watch* is a hierarchic state. It is defined by the HySChart in Figure 5. The labels refer to the following activities and actions:

$$t\_inc \equiv \dot{t} = 1$$

$$init \equiv v\_ok? \ \wedge \ v\_okk! \qquad late \equiv t = 45 \ \wedge \ v\_wrong!$$

$$early \equiv v\_ok? \ \wedge \ t <= 35 \ \wedge \ v\_wrong! \qquad T \equiv true$$

$$next \equiv v\_ok? \ \wedge \ t \in [35, 45] \ \wedge \ v\_okk! \qquad entry_{wait} \equiv t' = 0$$

The hierarchic activity *t_inc* states that t evolves in pace with global time in all (sub)states of the HySChart. The *Init* action performs the initialization, it is taken when the first *v_ok* signal is sent by *aVSyncR*. After that, state *wait* of the hierarchic state *watch* is entered. As long as no errors occur control remains inside the *watch* state and a *next*

transition is taken whenever the next *v_ok* signal arrives 35 to 45 ms after the preceding one. In this case a *v_okk* is sent to the video manager and the clock t is reset to 0 by the entry action of *wait*.

If no next *v_ok* signal arrives on time, the preemptive transition *late* is taken to state *error* and the signal *v_wrong* is emitted. The preemption concept used in HyCharts is weak preemption, i.e. all transitions within a hierarchic state are taken before the guards of the preemptive transitions (those originating from the hierarchic state and from none of its substates) are tested. Thus, a preemptive transition can be overwritten by a transition within the hierarchic state from which it originates. This concept is used here in order to ensure that the *late* transition can only be taken if no *v_ok* arrives within 45 ms after the preceding one: If a *v_ok* arrives exactly after 45 ms, the *next* transition within *watch* is taken and t is reset to 0 by the entry action of *wait*. After that the *late* transition is no longer possible since $t = 45$ does not hold.

The same concept is used in order to guarantee that the *early* transition, which is taken when a *v_ok* signal arrives too early, is not taken if a *v_ok* signal arrives exactly 35 ms after the preceding one. The initial state is *start* with $t = 0$.

Discussion. The semantics of HyCharts requires that the set of values for which a transition is enabled is (topologically) closed. Hence, we are not allowed to write $t < 35$ or $t > 45$ as guards of *early* and *late*, respectively. However, the video watchdog component shows how we can use preemption to work around this and still get the desired result.

The HySChart for the video watchdog component shows that (strict) timeouts can be modeled fairly easily in HyCharts. In contrast, considerable effort has to be spent in the timed automata model in order to make the transition which causes the timeout urgent, i.e. to ensure that it is taken as soon as it is enabled.

Note that priorities of transitions, as they are introduced by preemption, can also be used to model other variants of timeouts. For instance, on page 16 of [BFK+98] the authors mention that they actually wanted to demand that a *v_ok* signal occurs within the rightopen interval $[35\,\text{ms}, 45\,\text{ms})$. According to their explanation, a timed automaton which checks this timing cannot be specified with the timed automata variant supported by the UPPAAL model checker [LPY97]. Nevertheless, we can specify such behavior with HySCharts. Due to space limitations we cannot present the diagrams here, but the idea is as follows. We design an automaton with three levels of hierarchy. The top level is very similar to Figure 4, but does not cover the case where *v_ok* arrives too late. The second hierarchic level refines state *watch* and is largely similar to Figure 5. The third hierarchic level refines state *wait* and only contains a single state *wait_s* and a transition *late* leading from *wait_s* through the hierarchy levels to the *error* state. This transition overwrites the *next* transition at the second hierarchic level, if a *v_ok* arrives 45 ms after the preceding one. In a similar manner it is also possible to express *precise* timeouts, which demand that an event occurs exactly at time t, with HySCharts.

5 Conclusion

In this section we compare timed automata and HyCharts on basis of the model of the lip-sync protocol.

5.1 HyACharts

In Section 2 we saw that HyACharts are helpful in visualizing the structure of a system. The structure tells us which components communicate with each other, because system components can only interact via the channels depicted in the HyAChart. In contrast, timed automata do not have a concept of local signals or local variables. Every automaton may read and write any signal and variable. As a consequence, timed automata do not support modular specification (see [Sta97] for a similar result on hybrid automata). Visualization of the communication infrastructure between a set of parallel automata is not supported.

Sometimes it appears to be uncomfortable that a delay is required in each feedback loop in a HyAChart in order to ensure well-definedness. In practice it would be convenient if delays were added automatically by a CASE tool. However, the necessity of delays helps to keep implementation effects in mind. Real components can also only react with delay to their inputs. Thus, the engineer is forced to think about the effect a delay, as caused by implementation, has on his or her system. An alternative to demanding delays would be to employ a synchronous approach, like e.g. in Esterel, and perform causality checks of the specified design [Ber96].

5.2 HySCharts

In the lip-sync protocol the possibility to structure an automaton hierarchically was not fully exploited. Instead, hierarchy was mainly used in the HySCharts to specify common activities for sets of states and to specify priorities of transitions. In part this may be caused by the "flat" timed automata specification which was the starting point for this work. A development starting from the requirements may have resulted in a stronger hierarchic structure.

An important difference between HySCharts and timed automata is that transitions in HySCharts are always taken as soon as they are enabled, while transitions in timed automata can, but need not, be taken when they are enabled. Invariants are used in timed automata in order to enforce that a transition is taken. This has two important consequences:

- First, specifying non-determinism w.r.t. the time at which an enabled transition is taken is straightforward in timed automata. In HySCharts some extra thoughts are necessary to replace non-deterministically firing transitions by transitions depending on non-deterministically drifting clocks. Introducing macros to HySCharts, which allow the easy specification of non-determinism w.r.t. time, would be helpful here.

  However, the extra thoughts necessary for non-determinism w.r.t. time are compensated by the work saved when specifying urgent transitions. Urgent transitions are straightforward in HySCharts. In timed automata a suitable combination of invariants and transition guards is necessary to make transitions urgent. In the context of parallel automata which synchronize on the regarded transition, the situation becomes even more difficult as the invariants and transition guards of these parallel automata must also be considered. In the model of [BFK+98] a considerable amount

of work is spent on making transitions in the model urgent, even although UPPAAL offers specific features for urgent transitions.

Note that in the lip-sync case study non-determinism w.r.t. the time when a transition is taken is only needed to model the environment. The system itself always performs actions as soon as they are enabled.

- Second, the combination of transitions guards and invariants can easily lead to time deadlocks in timed automata, in particular in cases of many interacting parallel automata. In HySCharts time progress cannot be blocked, as there are no invariants and enabled transitions cannot be blocked by other parallel components. The price we pay to get rid of invariants and have urgent transitions is that the set for which a transition in a HySChart is enabled must be (topologically) closed. Closedness has to be required, because it ensures that there always is a well-defined least point in time for which the next transition is enabled (see the proof in [GSB98]). In the lip-sync protocol this closedness requirement prohibits to write timeout conditions like $t > 45\,\text{ms}$ directly. In this case study, however, closedness did not pose a problem. We used priorities of transitions to prevent unnecessary time-outs in cases in which the awaited signal arrives just within the required time bound $t \leq 45\,\text{ms}$. This use of priorities seems to be a standard technique.

A further characteristic of HyCharts, which did not appear in the diagrams given in this paper, is that HySCharts allow to send more than one signal (or event) in a single transition. In the underlying case study this is helpful for the *aInitSyncR* component. While eleven states are needed in the timed automata specification of it, two states suffice in the HySChart specification.

For both formalisms, HySCharts and timed automata, we note that automata fragments, or skeletons, which implement timers would have been useful in this case study, because timers are parts of various automata of the lip-sync protocol. In the case of HySCharts, the interface concept which underlies hierarchic states (see [GSB98]) might be employed to define such modular automata fragments.

5.3 Summary

The two most important differences between HyCharts and timed automata are the modularity offered by HyCharts, but not by timed automata, and the different concepts of when enabled transitions are taken, as outlined above.

Apart from that, many differences between timed automata and HyCharts are legitimated by the different aims of the description techniques. Timed automata specifically target at verification while HyCharts put their emphasis on convenient specification. Nevertheless, we think that the hierarchy and modularity of HyCharts enables efficient verification. Indeed there is research which tries to employ hierarchy for efficient model checking [AHM+98].

References

[ABSS96] A. Feyzi Ates, M. Bilgic, S. Saito, and B. Sarikaya. Using timed CSP for specification, verification and simulation of multimedia synchronization. *IEEE Journal on Selected Areas in Communications*, 14:126–137, 1996.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AHM+98] R. Alur, T.A. Henzinger, F.Y.C. Mang, S. Qadeer, S.K. Rajamani, and S. Tasiran. MOCHA: modularity in model checking. In A.J. Hu and M.Y. Vardi, editors, *CAV 98: Computer-aided Verification*, Lecture Notes in Computer Science 1427, pages 521–525. Springer-Verlag, 1998.

[BBBC98] G. Blair, L. Blair, H. Bowman, and A. Chetwynd. *Formal Specification of Distributed Multimedia Systems*. UCL Press, 1998.

[Ber96] G. Berry. The constructive semantics of Esterel. Book in preparation, 1996.

[BFK+98] 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.

[GS98] R. Grosu and T. Stauner. Modular and visual specification of hybrid systems – an introduction to HyCharts. Technical Report TUM-I9801, Technische Universität München, 1998.

[GSB98] 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.

[LPY97] 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.

[Reg93] T. Regan. Multimedia in temporal LOTOS: A lip synchronization algorithm. In *Proceedings of PSTV XIII, 13th Protocol Specification, Testing and Verification*, North-Holland, 1993.

[SGW94] B. Selic, G. Gullekson, and P. T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons Ltd, Chichester, 1994.

[SHH92] J.-B. Stefani, L. Hazard, and F. Horn. Computational model for distributed multimedia applications based on a synchronous programming language. *Computer Communications (Special Issue on FDTs)*, 15, 1992.

[Sta97] T. Stauner. Specification and verification of an electronic height control system using hybrid automata. Master's thesis, Technische Universität München, 1997.