Timed annotations with UML [*]

Susanne Graf and Ileana Ober

VERIMAG [**] – Centre Equation – 2, avenue de Vignate – F-38610 Gières – France
{Susanne.Graf,Ileana.Ober}@imag.fr
http://www-verimag.imag.fr/{graf,iober}

Abstract. In this paper we describe an approach for real-time modeling in UML focusing on analysis and verification of time related properties. As a motivation, we present some alternatives for extending an untimed computational model with time. We show that the use of time stamped events, representing instant of state changes, provides the right level of abstraction for reasoning about timed computations. This is also, at notation level, the choice of the OMG UML Real-Time Profile. We complete this profile by identifying important events and durations. The originality of our approach is that it provides a formal semantics of the time related primitives in terms of timed automata with urgency. An interesting point is that this time extension is defined independently of the semantics of the functional part.

1 Introduction

At a time when the real-time and embedded fields are using more and more modelling and verification, in particular time analysis and verification, we are surprised to notice the relatively little efforts on applying time analysis methods and verification to UML methods. The design and analysis of real-time and embedded systems relays on detailed knowledge not only of the functional requirements, but also of their real-time aspects. The time occupies a key place in the context of the real-time specification. The use of time-aware formalisms and the theoretical results on them, have made possible the development of model checking and verification tools: Kronos [Yov97] - based on timed automata [?],....

The purpose of this paper is to describe a framework for annotating UML models with time information in order to apply verification techniques on timed annotated models. This work was performed in the framework of the OMEGA project (http://www-omega.imag.fr). The purpose of this project is to develop a methodology for the development and specification of real-time and embedded systems that gets help from formal techniques in verification and model checking. In this context, we use a MDA approach for specifying real-time applications, by adding as much as possible information at model level. We think

[*] This work has been partially supported by the IST-2002-33522 OMEGA project

[**] VERIMAG is a research laboratory associated with CNRS, Université Joseph Fourier and Institut Nationale Polytechnique de Grenoble

here in particular at adding at model level platform dependent information, that is indispensable for deriving useful properties of the system.

Our approach consists in adding temporised annotations on a model in a systematic manner, and then performing model verification based on these properties (verify that they are not contradictory and infer additional properties). In order to do this, we define a set of possible time annotations based on events (see Section 2 for the advantages of an event driven approach) and inline with the UML RT Profile. We define the semantics of the timed part of a model using timed automata. This semantics is defined independently on the semantics of the functional part, and can be mixed with *any* semantics of the functional part provided that the events we use can be identified in that semantics.

The rest of the paper is organized as follows. Section 2 describe various strategies for adding time information into untimed specifications. Section 3 overviews related effort on working with time in UML models. Section 2 contains a discussion on the mechanisms for adding time information in an untimed framework. Section 4 presents the concrete approach in the context of UML, and Section 5 gives its semantics. The tool features based on this timed framework are described in Section 6. Concluding remarks and discussions are given in Section 7.

2 Extending computation with time

3 Related work on UML and time

UML offers a variety of notations for capturing many aspects of the software development, mostly focused on functional aspects, but also covering requirement analysis, implementation, verification, and testing. In this paper we address an often neglected aspect of UML: the specification of the model behavior with respect to time evolution. In this section we discuss the related work on explicit handling of time and/or of time related information.

The UML Real-Time profile [OMG02] (UML RT) represents the first step in answering OMG's request for a "*UML-based paradigm for modeling time-, schedulability-, and performance-related aspects of real-time system that would be used (among others) to (1) enable the construction of models that could be used to make quantitative predictions regarding these characteristics and (2) to enable inter operability between various analysis and design tools*". UML RT defines indeed a very rich vocabulary of real-time related concepts, and their mutual relationships. Due to its goals (mainly (2)), the UML RT remains at a high level it and does not enter into time semantic details. Although it represents a key step towards specifications completely covering the time dimension, as it is, the profile is not sufficient as it is for achieving this goal.

The UML 2.0 [Gro03] proposal (as far as available at the time we write these lines) gives more attention to time related aspects than the current UML standard [UML01]. Indeed, all the time-related concepts we added in Section 4 are already present in [Gro03] (therefore in the context of UML 2.0 we won't need to care for adding them explicitly). However, as the UML definition does not aim at providing a formal semantics to the concepts it defines, the time-related part contains open questions and does not offer a framework directly usable for simulation and verification.

In [Dou99] Douglass argues on the importance of specifying time-related information in (some) real-time systems. Douglass distinguishes between six kinds of time (absolute, mission, friendly, simulation, interval and duration), but most of these distinctions do not really help to solve the analysis problem. This approach can also be used for schedulability analysis. Douglass describes a timing analysis approach and tool that uses rate monotonic analysis (RMA) [HKO$^+$93] on a set of periodic tasks to (statistically) estimate execution times.

Lavazza et al. [LQV01] give an approach to real-time development centered on a concrete case study. The approach consists in translating UML models into first-order temporal logic with time, on which it uses model checking.

Knapp et al. [KMR02] use timed state machines and collaboration diagrams with time constraints, whose content is checked against the specification provided in the state machine. In order to do this, the authors compile the timed state machine into timed automata and they use the constraints contained in the collaboration diagrams to generate observer time automata. They are then used together with the model checker tool UPPAAL [LPY97] to verify the compatibility between constraints and specification.

On the side of commercial UML tools, we mention here Rhapsody [Ilo], who uses a synchronous approach. The system advances like a sequential system in global steps (cycles) and time is measured in number of cycles. The analysis of time within a cycle is RMA.

Although not UML-based, we mention here the related efforts done in the context of SDL [] because of the similarities of the two formalisms. SDL has basic concepts for: time, time-related data types (time and duration), and time measuring (timer). [BKM$^+$01] describes extensions needed to SDL to better address real-time needs. [Gra02] gives a framework for specifying real-time systems and performing verification on them. While, QSDL [**?**] propose to attach timing information by means of SDL constructs and to provide some minimal deployment information.

Several approaches dealing with *temporal* aspects on UML, basically handle *order* (in general preorder) of successive steps in the behaviour. This is the case for instance of the OCL extensions for specifying timing constraints [FM02]. Here, OCL is extended with logic, by adding operators for reasoning on state sequences.

Harel and al. [**?**] apply temporal logic on LSC [DH99], an extension of sequence diagrams with mandatory/optional behaviour. They embed a subset of LSC in CTL*, and can thus use LTL and CTL model checkers for verifying LSCs.

4 Framework for timed annotations

In this section we give a brief overview of the time constructs we use in our approach. We need time primitives (time related data types and time measuring mechanisms), that give us the vocabulary for the time annotations.

4.1 Time primitives

We use a very small set of time primitives, which are defined with the concern of being compatible with the UML RT [OMG02]. The time model is based on two data types: *Time* - relating to time instances, and *Duration* - relating to the time elapsed between two instances of time. A particular instance of time is *now*, which always holds the current time and it is visible throughout the whole model. *now* can be used in action specification, guards or constraints. Nevertheless, the model may not explicitly alter the value of *now*, this is done by some external uncontrolled mechanism, who satisfies the constraint that its values grow monotonically. Model dependent *time constraints* can add new restrictions on time progress, as discussed below. [1]

As in UML RT, we use two related timing mechanisms: *timer* and *clock*. *Timer* is a mechanism that generates *timeout* events when a specified duration time elapsed. A *clock* is similar to a periodic timer and emits *ticks*.

4.2 Timed events

We mentioned in Section 2 that we base our approach on events. In particular, in our setting all the events are timed, i.e. a time stamp holds their occurrence time.

An important dimension of our approach is the definition of a throughout *set of events* associated to all relevant moments of a behaviour. For instance, with any transmission of the signal *s* three events are associated: *send(s)* - the instant at which the signal is sent by the sender, *receive(s)* - the instant at which the signal reached the receiver (its input queue), and *consume(s)*. It may be that the three events have the same time stamp attached, or that the *send* and the *receive* have the same time (if there is no communication delay), however we need means to distinguish all these events.

Similarly, we associate events to *operation calls* (decomposed as two signal transmission, thus resulting in six events), *actions* (the initiation/termination of an action execution), *transitions* (the start/end of a transition), *states* (entry/exit in/of the state), *Boolean conditions* (the enabling/desabling of the condition), *timers and clocks* (the start/stop/reset of the timer, the timeout occur, the clock's tick, etc.).

[1] Note that UML we already have the case of an instance name that can be used all over the model without being defined. It is the case of *self* which refers to the runtime instance evaluating this expression.

The above described concepts: time-related data types, time measuring mechanisms and timed events (plus the means to identify them) represent the building blocks of our approach. They are used in the timed annotations.

4.3 Derived durations

The reasoning on the constraints between the occurrences of an event *ev1* and subsequent event *ev2* takes an important place in our approach. In a sequence of occurrences of *ev1* and *ev2*, several options exist to identify the "matching" (*ev1, ev2*) pairs. We distinguish between three patterns for matching pairs:

1. *duration*(*ev1, ev2*) represents the duration between an occurrence of *ev1* and the next occurrence of *ev2* such that there is no other occurrence of *ev1* in between. If evaluating *duration*(*ev1, ev2*) only at the occurrence of the first *ev2* after a (series of) *ev1*, the value of *duration(ev1, ev2)* is that of *now - time(ev1)* that is the time passed since the **last** event *ev1*. At other points of time, its value is difficult to express only in terms of occurrence times of events: $time(pre^n(ev2))$ - $time(ev1)$ where n is chosen such that $time(pre^{n-1}(ev2)) \leq time(ev1) \leq time(pre^n(ev2))$.
2. When several request events (each corresponding to *ev1*), may all be answered by a single *effect* event (corresponding to *ev2*), and this should happen within a limited time starting from the first request, the time elapsed between the **first** event of a series of occurrences of *ev1* and the first consecutive *ev2* is relevant.
3. Finally, we consider the case where several pairs *(ev1,ev2)* can be "active" simultaneously. In most cases, one can find a parameter x (or combination of parameters) identifying the matching pairs, and express the required constraint on matching pairs by *For any x:* $duration(ev1(*,x,*),ev2(*,x,*)) \leq d$. Nevertheless, due to the implicit use of FIFO buffers for storing signals in objects, it might be impossible to always find the matching parameters, and for this purpose we introduce *pipelineDuration*(*ev1, ev2*), where the matching pairs may be overlapping, observed in a pipelined fashion.

To facilitate the use of duration expressions, we define a set of duration expression corresponding to frequently used durations in practice, and we express them in terms of the predefined events mentioned above. For instance, we define the *client response time* as the time elapsed between the moment a call was emitted and the moment at which the response is consumed by the client. Additionally we define notions as *server response time*, *execution delay*, *execution time*, *period*, *reactivity*, and *transmission delay.* We do not give here the meaning of these predefined durations (for most of them the naming should be telling) nor the mapping to basic events (which is straightforward), for this the reader is referred to [SG02].

4.4 Timed annotations

The goal of our approach is to use timed annotations throughout a UML model, and to apply model checking and verification on these annotations. The timed annotations are time dependent Boolean expressions, that, depending on their use, can be interpreted in various ways: as *predicates*, as *invariants* or as *constraints*.

A *predicate* is a time dependent Boolean expression used as guard on state machine transitions or as conditions in decision actions. A time predicate is either *true* or *false* depending on the values of the instances (both time dependent and time independent) composing the Boolean expression. The time-dependent values are considered when the trigger event occurs.

An *invariant*, of the form $invariant(p)$ - where p is a time dependent predicate - is *true* only if p holds true on the entire execution leading to the state [2] to which it is attached. In order to ease the expression of invariants, we allow to "attach" invariants with states s or events e as a short hand for a global invariant of the form $invariant(in(s) \Rightarrow p)$ or $invariant(in(e) \Rightarrow p)$, meaning that p must hold at each occurrence of event e, respectively in all time points in which the system is in state s. Invariants can be used as *properties* which should be derivable by a given system description.

A *constraint* represents an invariant which is interpreted as assumed fact on the system under consideration, its environment or the underlying execution platform. An assignment $x := 0$ viewed as constraint can be read as *whenever this step is executed, in the state just after its execution, the variable x has the value* 0. Note that we don't have to care for how this is achieved exactly.

5 Semantics of the timed annotations

In this Section we give the basis of the semantics of the timed annotations introduced in Section 4. As anticipated in Section 2, our approach is based on events.

The semantics of the timed annotations is defined independently of the semantics of the functional behaviour. It is required that the untimed semantics can be viewed as a labeled transition systems where transitions represent events. Additionally, all the events referred to in the timed part must be identifiable in the transition system defining the functional semantics. This requirement is not very strong as most operational semantics use a state base approach where the needed events, or a subset of them, are identifiable. Note that if only a subset of these events can be identified in the untimed semantics, this only means that it is impossible to use the timed events that cannot be identified, while the use of the others is not restricted.

For instance, we have taken two different UML formal semantics on which we worked: one based on labeled transition systems [WD02] and the other written in ASM [Obe03]. In both of them the timed events mentioned in the previous section

[2] Here the *state* represents a state in the semantics, not one in a UML state machine. As we anticipated (and it is described more in detail in Section 5) the UML model execution can be regarded as a transition system, whom states we refer to and which are nit directly connected to state machine states. Note that even the execution of a UML model with no state machines results in states at the semantics level.

can be identified, though they were developed with no timming capabilities in mind, therefore it is possible to time both semantics using the approach described in this paper. Although the two semantics differ [3], both of them can be timed so that we can reason about e.g. the duration of an operation call or the execution time of an action.

The semantics of the time related concepts is defined by a set of timed automata with urgency [BST98] constraining the *occurrence time* (and only the occurrence time) of all the events whose possible occurrence ordering is given by the untimed semantics.

Thus, the *timed behaviour* of a system is the synchronized product of the event labeled transition system defining its functional behaviour and a set of time automata defining constraints on the occurrence times of events. The result is a global timed automaton defining constraints on the occurrence time of each event.

It is relatively easy to represent all the time related concepts described in Section 4 as timed automata. To illustrate our approach, we present here timed automaton associated with a timer.

The time automaton associated with a timer instance is given in Figure 1a. It does not express any constraint on the occurrence times of the *set* and *reset*, but we suppose that they occur immediately after the preceding action (in the same sequential behaviour), which in the timed automaton is expressed by an *eager* transition. Also, given an occurrence of *set*, this automaton constrains the occurrence time of the *timeout* (i.e. the *occur* event) to exactly the defined timeout time: *time(set(timer,delay)) + delay*. This also corresponds to an *eager* transition in the timed automaton.

The point of time of the consumption of the timeout, however, is not restricted by the timer itself. Additional constraints can be added in order to constrain the consumption of the timeout. For example *"always the time between the timeout and the corresponding consumption is smaller than d"*[4], which can be expressed by an invariant of the form;

$$\textit{invariant}(\ \textit{at}(\textit{consume}(t)) \Rightarrow \textit{now}\ \text{-}\ \textit{time}(\textit{occur}(t)) \leq \textit{consume-delay}\ )$$

or by the shorthand:

$$\textit{invariant}(\ \textit{duration}(\textit{occur}(t),\ \textit{consume}(t)) \leq \textit{consume-delay}\ )$$

Figure 1b shows the timed automaton corresponding to the composition of the timer timed automaton and the timed automata associated with this constraint, where the transition associated with the consume event is *delayable*, meaning that it will occur somewhere within the specified delay.

[3] For example, the concurrency model is different: while [WD02] considers that an active class has a single execution thread, in [Obe03] an active class may do several things at a time, a new thread being created for each accepted operation call.

[4] under the condition on the functional model that the timer is only set again after the timeout has been consumed

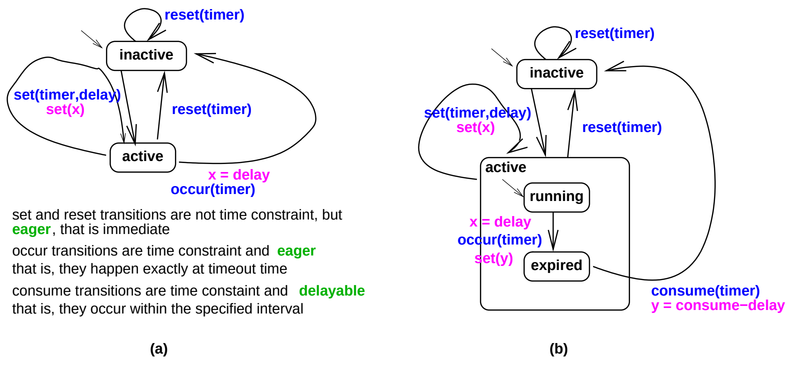
reset(timer)
inactive
set(timer,delay)
set(x)
reset(timer)
active
x = delay
occur(timer)
set and reset transitions are not time constraint, but eager, that is immediate
occur transitions are time constraint and eager that is, they happen exactly at timeout time
consume transitions are time constaint and delayable that is, they occur within the specified interval
(a)
reset(timer)
inactive
set(timer,delay)
set(x)
reset(timer)
active
running
x = delay
occur(timer)
set(y)
expired
consume(timer)
y = consume−delay
(b)

Fig. 1. timed automaton for a timer (a) with constraint consumption delay (b)

6 Tooling openings

The time related constructs described in Section 4 bring a benefit to the modeller only if included in some tool features. In this section we describe what tool features are made possible by the time constructs.

Our main concern is to express constraints on execution times and on durations between the defined events. For models where time is sensitive, the annotation with time constraints is needed. The annotation consists in giving both annotations corresponding to a priori information on the model (i.e. the assumption), and annotations corresponding to the timed requirements. These annotations can be given as constraints on the elements of a class diagram, of a statechart, or of an action specification.

Additionally, the time annotations can also be given on sequence diagrams. When given on sequence diagrams, the time annotations have a limited form: interval constraints on occurrence time of events or on durations between pairs of events and timers. This adds to the "classical" drawback of sequence diagrams that comes from the difficulty to interpret their meaning: for instance, consider an interval constraint between two events present in the sequence diagram, it is unclear whether it holds between any occurrences of the given events or not.

Such a UML model annotated with time information is fed to a timed verification tool. The verification tool can check that the various time annotations are not contradictory. If coupled with a simulator (i.e. symbolic execution) the time verification tool could filter the system runs that comply to all timed annotations, or detect that there is no such run.

The time annotations on the UML model may be *incomplete*, as opposed to complete specifications where on any piece of behaviour there is a, direct or indirect, time annotation (specified as constraint or requirement). Incomplete specifications make the verification problem more difficult.

In this case, the verification algorithm should *synthesize* for any transition the weakest constraint guaranteeing the satisfaction of all future constraints. When constraints are not “cyclically overlapping”, completing an incomplete sequential specification can be done in linear time, even when interval bounds have parameters. This is useful for simplifying simulation, even in the case where the overall system is infinite.

Nevertheless, constraint propagation can be done automatically with a reasonable effort only within a sequential behaviour- defined by a (small set of) active object(s). The budgeting over concurrent agents, must be provided by the user.

The correct specification of time information involves both knowledge on deployment and a good methodology for specifying this information. A good methodology for specifying time information is a specialization of some general UML modeling spiral methodology. At a first iteration some general time information is given, which is validated using the tools and refined or corrected in future steps, until we reach a stage where the timed annotations are detailed enough. In the framework of the IST project OMEGA we work on such a methodology.

An UML model with time annotations can be simulated (i.e. symbolically executed) using time aware simulators. Therefore, such a model can be used for model based testing. Obviously, the verification of arbitrarily complex properties is undecidable, due to the infiniteness of data domains, to the existence of unbounded signal queues and to dynamic object creation. Nevertheless, the verification of time related properties can often be done on a finite control abstraction, i.e. on a system with finite data domains, bounded signal queues and bounded object creation. A number of interesting verification problems are decidable on such a finite control abstraction under the condition that in the timed automata, obtained by translating the timed part, clocks can only be reset to zero, stopped and restarted, and the only allowed tests are comparisons of clock values or differences of clock values with constants. This means that in order to be able to effectively apply symbolic simulation and verification tools, at UML model level, the only allowed timed constraints imposed on events are Boolean combinations of comparisons of the type "*duration since the occurrence of some event lays within an interval*" or "*the difference of occurrence times between two past events lays within an interval*".

7 Conclusions

In this paper we describe an approach for adding time information to UML models. Our approach is based on the definition of a framework for timed annotations. This framework is compatible with the *UML Real-Time profile for Performance Scheduling and Real-Time* as it is based on some of the concepts defined by this profile. This framework also represents a specialization of the UML RT profile as it defines all the relationship between the considered concepts and it uses a set of events larger than those defined in this profile. *Events* take a central place in our approach which consists in using timed events as basis for all time annotations. In the discussion in Section 2 we show that the use of time stamped events, representing instant of state changes, provides the right level of abstraction for reasoning about timed computations.

The semantics of the time constraints and of the timed annotations is given in terms of timed automata with urgency that use the time stamped events described in Section 2. These events can in general be identified in any operational untimed semantics of UML. This is the case with the two UML semantics we have considered [WD02] and [Obe03]. The timed semantics of a UML model can be obtained by extending any UML semantics that can be viewed as a labelled transition system where the transitions represent the considered events (the two semantics mentioned above both satisfy this requirement).

Many advantages result from this modular definition of semantics. The first advantage is a result of the separation of concerns: we do not have to carry in the functional part time related information that is not relevant in this area, while on the timed part we do not have to take care of the semantics of all untimed constructs. This resulted semantics is easier to understand and to adapt to specific needs.

By the fact that we can use a stand alone UML semantics as basis for our UML timed semantics, we gain the throughout treatment of the untimed concepts. We consider that defining a semantics for timed UML models means not only to treat the timed part, but also to integrate in properly with the rest of UML concepts: inheritance, association, data structure etc. Nevertheless, this is the approach take until now when formalizing time in the context of UML. For instance, [KMR02] only considers time properties in the context of a small subset of the UML concepts (namely state machines) and does not take into account inheritance, data structure, etc, which are part of a regular UML model.

The time framework described in Section 4 and whose semantics is given in Section 5 serves as basis for tool features on time analysis. We present in Section 6 the tool features that can be developed in the context of this time framework. As part of the IST OMEGA project we intend to develop tools corresponding to these functionalities. The preliminary results obtained until now on small prototypes allow us to believe that this goal is realistic.

References

[BKM$^+$01] Marius Bozga, Susanne Graf Alain Kerbrat, Laurent Mounier, Iulian Ober, and Daniel Vincent. Timed extensions for sdl. In *SDL Forum 2001*, volume 2078 of *LNCS*. Springer Verlag, June 2001.

[BST98] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *International Symposium: Compositionality - The Significant Difference*, LNCS 1536, 1998.

[DH99] W. Damm and D. Harel. LSCs: Breathing life into Message Sequence Charts. In P. Ciancarini, A. Fantechi, and R. Gorrieri, editors, *FMOODS'99 IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems.* Kluwer Academic Publishers, 1999. Journal Version to appear in Journal on Formal Methods in System Design, July 2001.

[Dou99] Bruce Powel Douglass. *Doing Hard Time, Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns.* Object Technology Series. Addison-Wesley, 1999.

[FM02] Stephan Flake and Wolfgang Mueller. Specification of real-time properties for uml models. In *Proceedings of 35th HICSS*. IEEE, 2002.

[Gra02] Susanne Graf. Expression of time and duration constraints in sdl. In *3rd SAM Workshop on SDL and MSC, University of Wales Aberystwyth*, June 2002.

[Gro03] U2P Group. *UML 2.0 Superstructure proposal v.2.0.*, January 2003.

[HKO$^+$93] M.G. Harbour, M.H. Klein, R. Obenza, B. Pollak, and T. Ralya. *A Practitioner's Handbook for Real-Time Analysis.* Kluwer, 1993.

[Ilo] Ilogix. Rhapsody development environment.

[KMR02] Alexander Knapp, Stephan Merz, and Christopher Rauh. Model checking timed UML state machines and collaborations. In W. Damm and E.-R. Olderog, editors, *7th Intl. Symp. Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT 2002)*, volume 2469 of *Lecture Notes in Computer Science*, pages 395–414, Oldenburg, Germany, September 2002. Springer-Verlag.

[LPY97] K.G. Larsen, P. Petterson, and W. Yi. UPPAAL: Status & Developments. In O. Grumberg, editor, *Proceedings of CAV'97 (Haifa, Israel)*, volume 1254 of *LNCS*, pages 456–459. Springer, June 1997.

[LQV01] Luigi Lavazza, Gabriele Quaroni, and Matteo Venturelli. Combining uml and formal notions for modelling real-time systems. In *Joint 8th European Software Engineering Conference, 9th ACM SIGSOFT*. ACM SIGSOFT, 2001.

[Obe03] Ileana Ober. An ASM semantics for UML derived from the meta-model and incorporating actions. 2589, 2003.

[OMG02] OMG. Response to the OMG RFP for Schedulability, Performance and Time, v. 2.0. OMG ducument ad/2002-03-04, March 2002.

[SG02] Ileana Ober Susanne Graf. D112 - the omega time extensions. Technical report, 2002.

[UML01] *OMG Unified Modeling Language Specification,* Version 1.4, June 2001.

[WD02] Amir Pnueli Angelika Votintseva Werner Damm, Bernhard Josko. D.1.1.2 : A formal semantics for a uml kernel language. Technical report, 2002.

[Yov97] S. Yovine. KRONOS: A verification tool for real-time systems. *Springer International Journal of Software Tools for Technology Transfer*, 1(1-2), December 1997.