Using an UML profile for timing analysis with the IF validation tool-set*

Iulian Ober University of Toulouse-II ober@univ-tlse2.fr
Susanne Graf VERIMAG, Grenoble susanne.graf@imag.fr
Yuri Yushtein NLR, currently at CIMSOLUTIONS B.V, NL yushtein@xs4all.nl

Abstract: This paper shows on hand of a case study the usefulness of the UML profile with real-time defined in the Omega project and of the IF validation tool-set. The case study is about intricate timing aspects arising in a small but complex component of the airborne Medium Altitude Reconnaissance System produced by NLR[1].

The purpose is to show how automata-based timing analysis and verification tools can be used by field engineers for solving isolated hard points in a complex real-time design, even if the press-button verification of entire systems remains a remote goal.

We claim that the accessibility of such tools is largely improved by the use of a UML profile with intuitive features for modeling timing and related properties.

1 Introduction

Designing a real-time system, so as to satisfy all its real-time properties, leads often to complex verification problems. This complexity is due to the fact that time is intrinsically a *global* notion, implicitly aggregating the *relative* and *local* timing conditions appearing in system design.

For defining non-trivial systems, it is nevertheless mandatory to conceive them in terms of local hypotheses and solutions. Consequently, in a component-based approach, designers seem to be condemned to build systems by component aggregation, without knowing a priori what effect this aggregation will have on the timeliness of each component and of the system as a whole. Some relevant examples of unexpected timing conditions resulting from this aggregation will be shown on the case study presented in this paper.

A solution to this problem consists in using automated tools to analyze the timeliness of a subsystem. There are two large classes of methods: model-checking which analyzes a semantic model algorithmically, and theorem proving. This paper is mainly about using a model-checking approach, which is more automatic. Very high-level parametric models are sometimes tackled better by proof-based techniques; but in general these models are elaborated by the verification experts rather than the engineers and the distance between these high-level models and the developed system may be quite important. E.g. when using duration calculus [CHR92] as verification framework, almost everything of the functional model has to be abstracted.

Model-checking techniques can be more easily applied to models that are obtained by starting from a functional model of a system (which is a design artifact normally available for any system), and which can be enriched by adding timing relevant information. Such a model can be quite naturally obtained by the designer and provides a faithful representation of the system under development. It can be directly analyzed with an appropriate simulation tool.

Nevertheless, automated verification tools have well-known limitations, and a first obstacle for putting these tools effectively to work, is that the designers have to understand them and build the models having these limitations in mind. From our experience, interesting insights in the timing aspects of a system are usually gained only when the (unrelated) details of the functional part are abstracted away. This means that a model must be decomposed into small functionalities, which makes property depending abstractions more easy to construct and even to mechanize. In our experience, not many engineers do this naturally, but they can easily learn it when they see the benefit.

*This work has been partially funded by the European OMEGA project (IST-2001-33522).
[1]National Aerospace Laboratory, The Netherlands.

The second obstacle is the complexity of the formalism for capturing a timing model and its properties. A good formalism is one that is intuitive for the designers and based on concepts they are already using. In the literature there are various extensions of temporal logics with quantitative time operators, which have the required expressiveness. However, from our experience, property formalisms based on familiar concepts (like state machines) are more easily accepted by the users and are more expressive.

In this paper, we present the results of a case study conducted jointly by experts and industrial users, in which meaningful results about timing were obtained by analyzing a custom made model using a user friendly UML-based validation tool. The rest of the paper is structured as follows: §2 presents the case study, with focus on the timing aspects. §3 presents the approach and the model obtained for this case study using a specific formalism (the OMEGA UML profile), the main results of timing validation and the techniques employed in this experience. In §4 we discuss some conclusions that might be drawn from this study.

2 The MARS system

The acronym MARS stands for Medium Altitude Reconnaissance System. It controls a high resolution photo camera embedded in a military aircraft, taking pictures of the ground from medium altitude. The system counteracts the image quality degradation caused by the forward motion of the aircraft by creating a compensating motion of the film during the film exposure. The system is also responsible for annotating the frames with the current time and position. The system also performs health monitoring and alarm processing functions.

Exposure control (Forward Motion Compensation and Frame Rate) as well as annotations are being computed in real-time based on the current aircraft altitude, ground speed, navigation data (latitude, longitude, heading), time-of-day, etc. These parameters are acquired from the avionics data bus of the aircraft.

2.1 The Databus Manager

For the purpose of this case study we concentrated on a sub-system which presents interesting timing problems, called Databus Manager (DM); it monitors the health of the data bus controller and, in general, of any communication going trough the data bus.

The system receives *data* concerning altitude and navigation from other components of the avionic system. The DM component supervises the reception of data messages and provides a *status* used by the system's alarm logic. In addition, the DM periodically polls the databus controller status and changes its own status accordingly to *Operational*, *BusError* or *ControllerError*. The precise requirements on the DM status computation are described below.

The two types of *data* inputs of the DM are received periodically, with some period ($P = 25ms$ in the concrete example) and jitter ($\pm J = 5ms$), and may occasionally get lost. The periods are not synchronized and may have any offset (smaller than the period). Figure 1 shows a possible configuration of the reception windows along the time axis (windows in which no message reaches the DM are marked with KO). The basic functional requirements on the DM status are:

- Controller failure leads to a change of status to *ControllerError*. Recovery leads to *BusError*.
- Status changes from *BusError* to *Operational* when two consecutive messages are (correctly) received from both sources (assuming no controller error).
- Status changes from *Operational* to *BusError* when three consecutive messages from a source are lost.

Note that these requirements do not define *when* the status change must take place. In fact, maximal reactivity is desirable. Two reactivity measures (at least) can be defined:

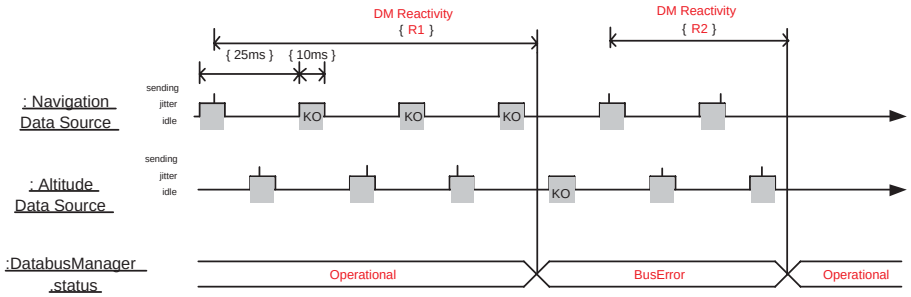

Figure 1: Timing of the message reception windows, DM status and reactivity measures.

- reactivity to losses, defined as the upper bound that DM guarantees for the time ($R1$) between the last correctly received message from the source causing a switch to *BusError*, and the actual moment of the switch.

  Simple problem analysis shows that the optimal $R1$ is $85ms$ ($3P + 2J$), in the case of fixed communication time between sender and receiver. This is the ideal reactivity that the DM design should try to approach.

- reactivity to recovery, defined as the upper bound that DM guarantees for the time ($R2$) between the first message in a series of correct messages leading to a switch to *Operational*, and the actual moment of the switch.

  In this case, the optimal $R2$ depends on the offset between the periods of the two data sources. However, even in the worst case, it is less than $60ms$ ($2P + 2J$).

Our experiments are described in the next section. They had two goals: (1) to check that the proposed designs verify the above mentioned functional properties, and (2) to determine the reactivity bounds offered by the different proposed designs (and point out the optimal solution).

3 UML modeling and validation experiments

3.1 Background on the OMEGA profile and the IFx toolset

The MARS sub-system was modeled using the OMEGA UML profile and timing and functional validation was performed using the IFx toolset. Here, we briefly introduce these technologies.

The OMEGA profile defines an operational semantics for a subset of UML, designed to suit the needs of designers of real-time embedded systems. On the *functional* side, the semantics defines aspects pertaining to control (like the rules governing concurrency) and communication primitives. These aspects are handled similarly as in the profile of the Rhapsody tool, and they are detailed in [DJPV03, DJPV05].

The *timing* aspects are described in detail in [GOO05]. The profile is compatible with the basic time related notions of UML 2.0 by defining a series of lightweight extensions to UML for describing time-driven behavior using timers, clocks, and timed guards. In addition, it allows to define transition urgency for categories of transitions[2]. For the expression of timing and functional requirements, the OMEGA UML proposes to use a notion of *events* and *observer objects*. Events are any semantic level state changes, similar as the notion of *timed event* in the SPT profile [OMG02]; in order to make this notion concrete, a notation has been defined for being able to name all semantic events by referring to a syntactic entity. Observers are characterized by a state machine which reacts to (semantic level) events and conditions occurring in the

[2] this concept is taken from timed automata with urgency [BS97]

system, and which acts as an acceptor of system executions – by use of states stereotyped with <<error>> as final states. An example of a property expressed by an observer can be seen in figure 5.

IFx [OGO05] is a toolset providing extended simulation and verification functionalities for OMEGA UML models. The core of the tool is a state space exploration engine for systems consisting of extended communicating timed automata (IF [BGM02, BGO+04]). In order to scale to complex models, IF provides several optimizations and and supports abstraction. The tool implements static and dynamic optimizations like dead variable factorization, dead code elimination, partial-order reduction and abstract interpretation of clocks. All optimizations strongly preserve timed safety properties which are of interest in the MARS system. In addition, the tool supports simple abstractions which preserve satisfaction of safety properties, but may show spurious counter-examples.

3.2 Overview on the UML model for MARS

The architecture of the MARS model as proposed by the designer of the system, is shown in the UML context diagram in Figure 2[3]. The main component is the *DatabusManager* object which maintains the global status and monitors message loss. For simplicity, the designer has separated the polling of the bus controller in a different object, the *ControllerMonitor*.

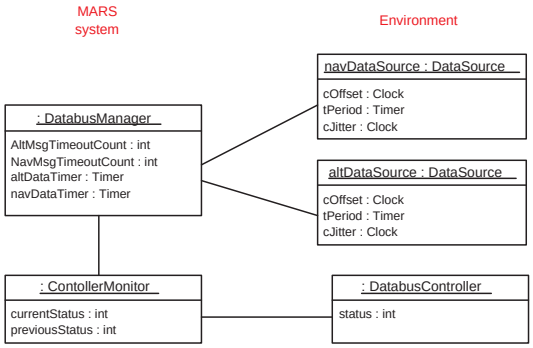

Figure 2: Structure of the MARS model.

In order to verify the DM under the assumptions on message arrival and controller errors mentioned in §2.1, the OMEGA profile allows to model the environment using the same concepts as for modeling the system, in particular an explicit object with the behavior expressed by the assumption. In Figure 2, we see therefore three environment objects corresponding to the altitude data source, the navigation data source and the bus controller.

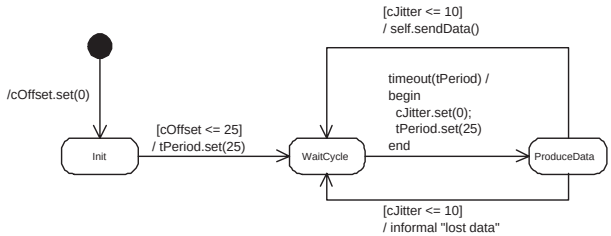

Figure 3: Environment model : state machine of the data sources.

In particular for modeling the environment, the possibility to express nondeterministic behavior is important. This is allowed in the Omega profile. For example, Figure 3 shows the state machine of data sources,

[3] associations express here the fact that the corresponding objects may communicate through signals or method calls

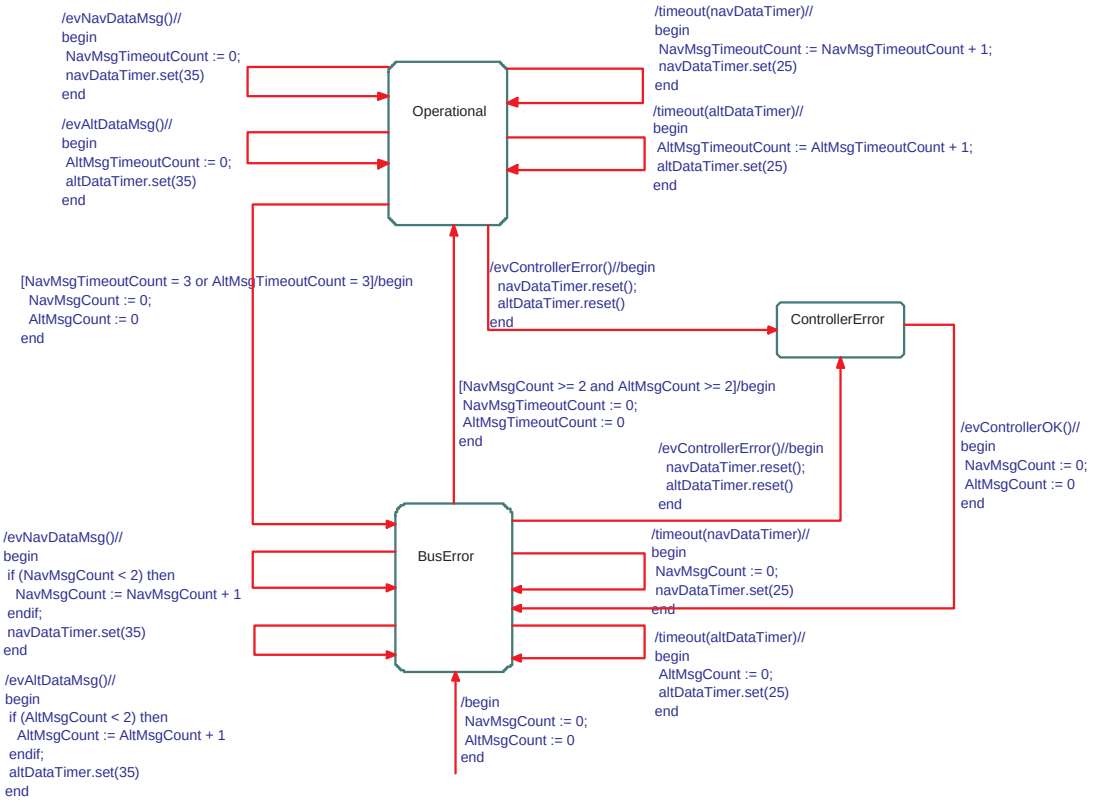

Figure 4: State machine of the DatabusManager.

using interval conditions on clocks to model the nondeterminism introduced by the starting time and by jitter. This state-machine indeed describes a data source with the required period and jitter as all transitions of environment objects are interpreted as *delayable*[4], that is, once they are enabled, they will be taken before their time guard becomes false or they may be disabled by some discrete transition. Moreover, Zeno computations[5] are not valid computations which guarantees in this example that the computation cannot "get stuck" in any state.

As we will see later, the requirements concerning the DM can be achieved in several ways. The first design provided by the engineer, shown in Figure 4, consisted of a single state machine with transitions triggered by events from both data sources ($evAltDataMsg$, $evNavDataMsg$) and from the *ControllerMonitor* ($evControllerError$, $evControllerOK$), or by timeouts corresponding to message loss detection ($altDataTimer$, $navDataTimer$). The principle is to use a timer to measure the duration of the period for each Data source — where the end of the period is defined by the reception of a message, where the timer is rearmed, or a timeout — and to always keep track of the number of consecutive received, respectively lost, messages. Notice that in this design we have chosen system transitions to be *eager*, that is they do occur at the earliest point of time at which they are enabled[6]. must progress,

[4]according to the terminology defined in timed automata with urgency [BS97]

[5]infinite computations with finite time progress

[6]Note also, that transitions are event triggered rather than time triggered as *timeout* is the event associated with a time condition. This is equivalent, but closer to operational way of thinking of the designers

3.3 Expressing properties and first evaluation results

Both, the functional and the reactivity properties described in § 2.1 can be expressed as observers to be verified on this model. Figure 5 shows the observer checking a bound guaranteed for $R1$, that is the maximal delay needed for transmission problem detection (see section 2.1). Note that observer transitions *synchronize* with observed events, and thus take place at the same time point as the observed event. Note also that this observer monitors only one data source (the altitude data source); we argue that the failure of the other source can only bring the DM into the $BusError$ status earlier, thus the maximum value for $R1$ is exposed when the other source does not fail (or it's failure is not observed).

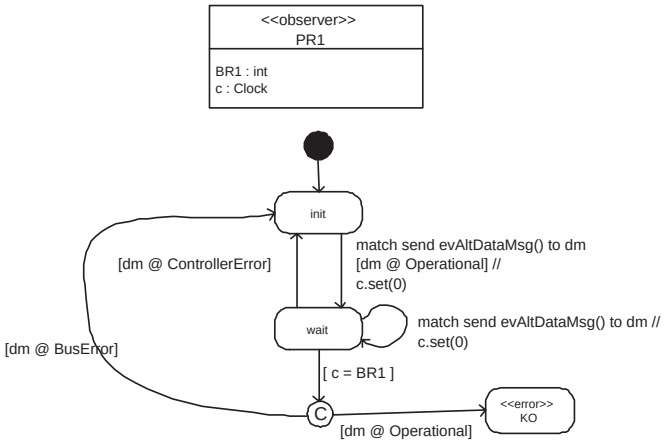

Figure 5: Observer for verifying the reactivity bound for $R1$.

All functional and reactivity properties were verified against the initial design presented above.

Due to state explosion problems encountered, a simplifying assumption was made first on the environment: the cycles of the two data sources are synchronized (i.e., their periods start always at the same time; their data may nevertheless be sent at different moments due to jitter). This assumption is not conservative[7] as the reaction time to message loss may (and turn out to) be longer when the two sources are not synchronized. In order to fully verify the properties, a different model, which is a conservative abstraction, is presented later on.

Nevertheless, this initial model was useful for understanding the potential problems, for debugging the model and ruling out some variants that had been proposed to increase efficiency. Under the simplifying assumption, all functional properties have finally been proved to hold on debugged versions of the DM design. An interesting outcome is that very similar designs may present different reactivity bounds.

For example, consider a slight variant of the design model presented above, in which in the *operational* status a *long timeout* is used, detecting the absence of messages during three consecutive periods, instead of detecting absence of individual messages and counting them. At a first sight, this new version looks more efficient as it does not need counter when the status is *operational*.

Using different variants of the reactivity constraints defined by the observer in Figure 5, we have determined with the help of our verification tool that the initial design has a better reactivity (85ms[8]) than the new one (only 110ms). As the motivation for the entire case study was to gain reactivity with respect to the existing synchronous design, which observes events only at fixed time points, this is not an acceptable solution.

The diagnostic traces provided by the model checker show that the difference stems from the way timers are handled at the transition from the *BusError* to the *Operational* status: in the initial version, timers are not affected by a status change, they just count periods of the data sources, while in the new version, the

[7] it leads to a simplified model that allows to find bugs, but we are not allowed to deduce correctness of the initial model from the fact that this model does satisfy some property

[8] that means it achieves the optimal reactivity as explained in section 2.1

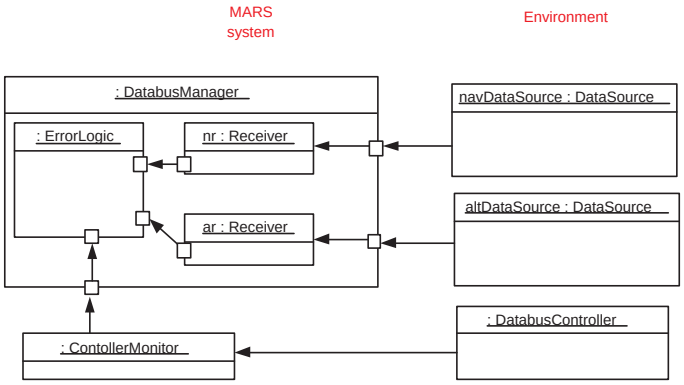

Figure 6: Decomposition of the DM.

long timer is not needed in *BusError* status and initialized when entering the operational mode. This might delay the detection of a bus error occurring right at this moment by an entire period. In order to correct the problem, the timer must be set depending on the actual “age” of the ”period timer” which is used when the status is *BusError*.

This kind of errors is quite common when trying to optimize a design, and the tools were very helpful by checking after any modification all the properties. This allows to get immediate feedback as running the verification is generally really fast (see also the table at the end of the section). Another group has used in parallel a tool based automatic or interactive theorem proving. They could prove at the end a parametric version of the property which is impossible to prove with a model-checker like ours, but they were extremely thankful to the extremely quick feedback provided by our tool: once we had a (simplified) working model on which the properties can be shown to hold, one can analyze small modifications of the UML model, just by “pushing the button” to translate the model and then run the verification of all properties, or “find a bug”.

3.4 Use of a compositional model and abstractions

In order to fully verify the desired properties without making the (unrealistic) assumption that both data sources are synchronized, we have proposed to use a model of the DM that is itself a composition of smaller entities (see Figure 6), in particular

- A *Receiver* component for each data source; it supervises the messages of a single source and keeps track of the message status in the last 3 windows and sends this status by means of *evCnt* messages at the end of each period.
- An *ErrorLogic* component which, based on the *evCnt* messages from the different *Receivers*, defines the status. The status changes from *BusError* to *Operational* when all *Receivers* have received correct messages during the last 2 windows, and from *Operational* to *BusError* when at least one *Receiver* has not received any correct message during the last 3 windows.

Using this model, we could verify all properties — or rather adaptations of them — using a compositional and conservative abstraction.

The abstraction used consists in replacing one *Receiver* with a chaotic abstraction *ReceiverAbs* which may send *evCnt* with any parameter at any time. This is a very rough over-approximation of the source–receiver pair, but it proved to be sufficient for preserving the desired properties. The abstraction is particularly interesting as it represents an over approximation of an arbitrary number of data receivers, meaning that it allows to verify the DM for the case with an arbitrary number of such data sources[9].

[9] always under the hypothesis that the time for taking decisions remains “negligible”

Configuration	Number of states	Number of transitions	User time
Initial model with only one source (no *CM* polling) (*non-conservative*)	1084	1420	$< 1s$
Initial model with two synchronized sources (no *CM* polling, *non-conservative*)	99355	151926	$36s$
Initial model with two de-synchronized sources (no *CM* polling) (*conservative* – exploration doesn't terminate)	> 1136768	> 1676126	$> 9m30s$
Abstract model, 10ms *CM* polling (*conservative* – does not terminate)	> 1494864	> 701120	$> 8m12$
Abstract model (no *CM* polling) (*non-conservative*)	118690	174871	$45ms$
Abstract model with non-det. *CM* polling (*conservative*)	155166	263368	$1m21s$

Figure 7: Verification times and state spaces for different verification configurations.

A second conservative abstraction used consists in replacing the deterministic polling cycle of the *ControllerMonitor* (10ms in the initial model) by a completely non-deterministic polling policy. While this introduces new executions, impossible in the initial model, the resulting state space is smaller as many previously disjoint states are grouped together[10].

The table in Figure 7 below shows the size of the state space and the processing time for several configurations of the MARS system which allows to draw some conclusions on the efficiency of the use of compositional models and in particular compositional abstractions. In particular, desynchronizing two resources has a tremendous effect on the size of the state space which is in fact due to the simultaneous presence of jitter and desynchronization. Notice that the effect is much more important than it looks like as we have stopped the exploration when, after reaching 1 mio states and 10 minutes, the state space was still growing rapidly — experience told us that it was likely not to be worth to wait until reaching 10 mio states; even if it would converge by then, what we didn't anticipate really, the result was not very useful for us, as we wanted to be able to rerun experiences on variants in short time. As we have not run our experiences on particularly well equipped machine (especially in memory), this means that we can still gain a few orders of magnitude and handle slightly more complex systems.

The use of both types of conservative abstractions leads to a state space of about the same size as the much simpler system with synchronized sources, which is still precise enough to satisfy all properties.

4 Conclusion

By using a case study as support, we have shown both, the convenience of the OMEGA UML profile for the expression of timed models and timed properties and the usefulness of the IF front-end for UML which allows for both flexible interactive simulation and complete state space exploration for debugging and verification of UML models.

We believe that the experiment presented shows that timing analysis tools can be used efficiently for solving isolated, hard timing problems in a UML design, even if fully automated verification for large designs remains a remote goal. Also we believe that more systematic use of functional decomposition as used in the example, can definitively help to make possible the verification of much larger designs in a compositional fashion, as there is no need for the verification of a model in which all parts are described in all details.

[10] but only if a symbolic representation of time constraints is used

The use of the OMEGA UML profile to capture timing properties has favored a very quick learning and adoption of our tools by experienced UML designers. Without the knowledge of a verification expert, the designers were able to use even advanced techniques like abstractions.

The relaxation of timing constraints — such as the abstraction from the polling period in the example — shows to be a very efficient abstraction technique in such models, and it is usually very simple to model. This kind of abstractions is always conservative for the satisfaction of (timed) safety properties. On the other hand, can introduce spurious error traces. However, in the MARS example this has never occurred, showing first, that with some exercise, a designer can learn to use abstractions which do not break the verified properties. And second, that the designers tend in the first place to build over constrained models. The reason is probably that they are strongly influenced by the requirement that programs must be deterministic, and they apply this also to specifications even if this is not needed for satisfying the requirements.

We have also found out during the experiments that some methodological guidelines for writing observers and for using the IFx toolbox are necessary during the learning process. A set of guidelines has been developed as a side result of this teamwork (see also [OGL05]).

References

[BGM02] Marius Bozga, Susanne Graf, and L. Mounier. IF-2.0: A Validation Environment for Component-Based Real-Time Systems. In *Proceedings of Conference on Computer Aided Verification, CAV'02, Copenhagen*, number 2404 in LNCS. Springer Verlag, June 2002.

[BGO+04] Marius Bozga, Susanne Graf, Ileana Ober, Iulian Ober, and Joseph Sifakis. The IF toolset. In *SFM-04:RT 4th Int. School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time*, number 3185 in LNCS, June 2004.

[BS97] S. Bornot and J. Sifakis. Relating Time Progress and Deadlines in Hybrid Systems. In *International Workshop, HART'97, Grenoble*, LNCS 1201, pages 286–300. Spinger Verlag, March 1997.

[CHR92] Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A Calculus of Durations. *Information Processing Letters*, 40(5):269–276, 1992.

[DJPV03] Werner Damm, Bernhard Josko, Amir Pnueli, and Angelika Votintseva. Understanding UML: A Formal Semantics of Concurrency and Communication in Real-Time UML. In Frank de Boer, Marcello Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Proceedings of the 1st Symposium on Formal Methods for Components and Objects (FMCO 2002)*, volume 2852 of *LNCS Tutorials*, pages 70–98, 2003.

[DJPV05] Werner Damm, Bernhard Josko, Amir Pnueli, and Angelika Votintseva. A discrete-time UML semantics for concurrency and communication in safety-critical applications. *Science of Computer Programming*, 2005. (to appear).

[GOO05] Susanne Graf, Ileana Ober, and Iulian Ober. Timed annotations in UML. *STTT, Int. Journal on Software Tools for Technology Transfer*, 2005. under press.

[OGL05] Iulian Ober, Susanne Graf, and David Lessens. A case study in UML model-based dynamic validation: the Ariane-5 launcher software. submitted, 2005.

[OGO05] Iulian Ober, Susanne Graf, and Ileana Ober. Validating timed UML models by simulation and verification. *STTT, Int. Journal on Software Tools for Technology Transfer, 2004*, 2005. Under press.

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