Modelling: Model Composition

KUN

August 2007 (revised version)

AMETIST DELIVERABLE 1.2

Project acronym: AMETIST
Project full title: Advanced Methods for Timed Systems
Project no.: IST-2001-35304
Project Co-ordinator: Frits Vaandrager
Project Start Date: 1 April 02
Duration: 36 months
Project home page: `http://ametist.cs.utwente.nl/`

1 Introduction

This deliverable presents an overview of the work done within Task 1.2 of the AMETIST project. To model larger system and control problems effectively one needs various mechanisms for constructing systems from subsystems. Within Task 1.2, the AMETIST consortium has studied how important, well-understood compositional methods from various areas including I/O automata, UML, and live sequence charts extend to control synthesis and verification problems in the timed automata setting.

The related IST project OMEGA aimed at providing a formal foundation for the unambiguous description of real-time, reactive, embedded systems in UML. Three partners from AMETIST (VERIMAG, KUN, WIS) also participated in the OMEGA project, which naturally led to close links between the two projects, in particular within Task 1.2. The two projects have made significant contributions to four areas: (a) urgency and timed I/O automata, (b) the IF language, (c) semantics of (real-time) UML, (d) scenario based specification. In the next sections, we describe these contributions in more detail, and try to draw some conclusions. Since several of the involved researchers participated both in OMEGA and in AMETIST, it is sometimes hard to attribute results to one specific project, but nevertheless this is what we will do for each of the four areas.

2 Urgency and Timed I/O Automata

Timed automata are very expressive and permit one to describe systems in which time stops and no further transitions are possible, so called *time deadlocks*. Unless one aims at modeling black holes or other physical singularities, this type of behavior should be considered as a serious modelling mistake. In timed automata, time is a global variable and if one component has a time deadlock, as a result the whole system may end up in a time deadlock as well. This means that a system may satisfy certain safety properties (e.g., that a bad state cannot be reached) simply because a time deadlock in one component cuts of whole parts of the state space. Hence, one can not trust the conclusions of model checkers in the presence of time deadlocks! Absence of time deadlocks is in general not preserved by model composition: if we compose two models without time deadlocks, the resulting model may exhibit a time deadlock. As a consequence, time deadlocks are not easy to detect for networks of timed automata, and require a full exploration of the state space.

In an attempt to address this problem, Sifakis and his colleagues [4, 28] advocate the use of *deadlines* for the specification of progress properties. Rather than using location invariants (as in UPPAAL), each transition of a timed automaton is decorated with an additional deadline predicate, which specifies when the transition becomes urgent. An advantage of deadline approach is that under some reasonable assumptions, it ensures what is called *time reactivity* in [4] and *timelock freedom* in [5], that is, whenever time progress stops there exists at least one enabled transition. Clearly, time reactivity implies absence of time deadlocks. Under certain conditions, time reactivity is even preserved by

parallel composition of automata [5, 4, 3]. The notion of deadlines has been incorporated in the IF toolset [6] and in MoDeST [9].

Deadline predicates are more difficult to implement in model checkers than invariants because they easily lead to non-convex zones. Non-convex zones indeed arise in the implementation of timed automata with deadlines in the IF toolset [6]. In particular, time transitions may lead from one convex zone to several convex zones (not only one, as in standard timed automata with invariants). When such a situation arises in IF, the non-convex zone is automatically split in several, possibly overlapping, convex zones. Due to the convexity problems, and the resulting loss of performance, deadline predicates have thus far not been implemented in UPPAAL.

As a contribtion to Task 1.2, the work of [10] proposes a small variation of deadline predicates, called *urgency predicates*. Informally, if for a certain transition both the guard and the urgency predicate evaluate to true, time may not pass any further and a transition has to be taken immediately. A small but significant difference with the approach of Sifakis et al [28, 4] is that Sifakis et al require that a deadline predicate *implies* the precondition predicate, whereas [10] achieves a similar effect by *conjoining* the urgency predicate with the precondition. In practice the use of urgency predicates leads to slightly shorter and more natural specifications than any of the other approaches. In order to ensure that time reactivity is preserved by model composition, [10] imposes the input/output distinction of the I/O automata model, including the requirements that (a) input actions are always enabled, and (b) different components never share output actions. In addition, some natural syntactic conditions are identified which ensure that convexity of zones is preserved.

Time reactive models may still exhibit undesirable behavior in which the system continues to perform transitions but time may not advance beyond a certain bound, so called *Zeno behavior*. Absence of Zeno behavior is in general not preserved by model composition: if we compose two models without Zeno behavior the resulting model may exhibit Zeno behavior. (Think of a system that immediately produces an output on port Q whenever it receives an input on port P, composed with a system that immediately produces an output on port P initially and whenever it receives an input on port Q.) Zeno behavior should be considered as a serious specification error, since one can not trust the outcomes of model checkers in their presence.

As a contribution to Task 1.2, the work of [22, 23, 24] (carried out in close collaboration with MIT) addresses the Zeno behavior issue within the general framework of *timed I/O automata (TIOAs)*. Whereas most emphasis within AMETIST has been put on efficient automatic verification, which implies limits on expressivity, the TIOA work focuses on expressivity and deductive verification. In [22, 23, 24], a class of *receptive* TIOAs is identified. Informally, a timed automaton is receptive if it has a strategy to let time diverge, irrespective of the inputs provided by the environment. Receptivity has the following nice properties: (a) for individual components receptivity can be proved easily, (b) receptive TIOAs are time reactive by definition, (c) receptive TIOAs do not exhibit Zeno behavior, and (d) the class of receptive TIOAs is closed under model composition. A beta release of Tempo, a toolset built by VeroModo Inc that supports the TIOA framework and includes

a translation to UPPAAL, has become recently available at `http://www.veromodo.com/`.

2.1 Conclusions

Time deadlocks and Zeno behaviors constitute slightly annoying anomalies of the timed automata framework, comparable to the division by zero problem in arithmetic. In general, time deadlocks and Zeno behaviors are hard to detect, and therefore automatic or methodological support to avoid their occurrence is highly desirable. In the IF toolset, time deadlocks are eliminated at the price of a small performance loss (non convex zones). In the UPPAAL toolset the issues are thus far simply ignored. The AMETIST results of [10, 22, 23, 24] provide a general method for dealing with both time deadlocks and Zeno behavior. It is shown that time reactivity (and hence absence of time deadlocks) can be guaranteed without any performance loss at the price of some reasonable syntactic restrictions. Users may apply the approach to manually prove time reactivity and absence of Zeno behavior for their models (for example applications see [24, 29]). More work is needed to fully integrate these results within timed automata model checkers and to mechanize receptivity checking.

We believe that establishing strong links between the TIOA framework and timed automata tools will help to combine the benefits of deductive verification and model checking.

3 The IF Language

A very interesting approach for increasing the expressivity of timed automata has been chosen in the context of the IF language and toolset [7].

The IF (= Intermediate Format) common modeling language allows the description of systems consisting of processes running in parallel and communicating through message passing via communication buffers, through signal exchange or through shared variables. Each process may use several clocks to measure time and transitions may be guarded by timing constraints and decorated with explicit deadlines. The language provide several type constructors such as enumeration, range, array, record, abstract as well as predefined basic types in order to simplify complex data description and manipulation. The language includes dynamic creation and destruction of process and signal route (channel) instances. This makes system configuration to be dynamic, that is, the number of components running (and in turn, the number of clocks ...) may change during execution. The common language integrates hierarchical states (to structure automata) and composed transitions basic control statements such as if-then-else and while-do are provided to structure automata transitions

The IF toolset supports different kinds of verification techniques, based on the enumeration of reachable states, such as on-the-fly verification of safety properties, generation of complete state-graphs, which can then be reduced modulo bisimulation. This second approach can be used to synthesise properties rather than verifying them, and also for compositional

verification. Time constraints can be either represented symbolically as in the tools Uppaal and Kronos - in this case with each discrete state a time zone represented by a DBM is associated - or in a discrete manner - such that clocks are represented by bounded integer having a particular value at any time.

Obviously, in this framework decidability is lost, but it is possible to directly map high-level languages such as SDL or UML into IF by preserving concepts which are useful for efficient validation. The effort for providing abstraction and other reduction methods applied to structured models (e.g. those based on static analysis and partial order reduction) can be developed independently of the user language. Obviously, this is only possible, if not too many concepts are lost by the translation. One aim of the work on IF was to provide "the right concepts" and this is still ongoing effort. For example, the scheduling problem provided by the case study of Axxom, could be modelled in IF by representing shared resources by processes and by using guards to restrict the search space [8]. As the notion of resource is general concept and priorities constitute a general means to express scheduling constraints, they have been integrated, in the context of OMEGA, into IF.

3.1 Conclusions

The work on the IF language and toolset has not been supported by AMETIST, but was funded by OMEGA and other projects. However, through the participation of one of the IF designers in AMETIST, we have been able to closely follow the developments, and, as explained above, work on the AMETIST case studies has influenced the development of IF. A number of observations can be made:

- Due to its nice GUI and its emphasis on performance, and investments in tool support, the numbers of users and applications of UPPAAL are currently much bigger than for IF. The role of IF is primarily restricted to that of a research vehicle.
- Several features of IF have been geared towards applications in the telecom area and towards UML (message passing, signals,..) and are probably less useful for dealing with the type of case studies purued within AMETIST.
- Dynamic creation and destruction of processes, however, appear to be a composition constructs that are very useful/natural in almost any application domain. For instance, in value chain problems it is natural to create a new process whenever a new job arrives. Adding process creation leads to challenging verification problems (infinite state and, in general, undecidable) and complicates the representation of states in model checking. Nevertheless, if the ambition is to turn UPPAALinto a mature, industrial tool, it should be extended with process creation along the lines of IF.

4 Semantics of (Real-Time) UML

During the second year of AMETIST, we studied the meaning of basic class diagrams in which the behavior of objects is described by timed state machines [21, 20]. These timed state machines are obtained by adding the basic vocabulary of timed automata (clock variables, guards, location invariants, clock resets) to UML state machines. Reactive objects may communicate by means of asynchronous signals and synchronous operation calls. The resulting semantics was defined in the typed logic of the interactive theorem prover PVS, and serves as a basis for a translation into timed automata. Although the main ideas about the intended semantics were rather clear, it turned out to be far from trivial to make this precise, and a large number of issues about inheritance, control, primitive and triggered operations, and signals had to be resolved. In subsequent work, we have used tlpvs, our PVS-based implementation of linear temporal logic and some of its proof rules, to verify two examples [1] (a fragment of the Medium Altitude Reconnaissance System (MARS) and the Sieve of Erathosthenes). Also, we have made our semantics compositional to support compositional reasoning [26].

An alternative approach for handling Real-time UML specifications by translating them component wise into timed automata, has been explored in OMEGA and published in [27]. In this work, most of the structural and behavioural characteristics of classes and their interplay are taken into account and issues like the combination of operations, state machines, inheritance and polymorphism are tackled, in the context of a particular semantic profile for communication and concurrency. The UML dialect considered in OMEGA, also includes a set of extensions for expressing timing. Our approach is implemented by a tool importing UML models via an XMI repository, and thus supporting several commercial and non-commercial UML editors. In this work we had two complementary aims. First, provide a means for efficient analysis and verification based on model exploration, where we mainly focus on time (and scheduling) related properties. This is achieved by a translation into the format of the powerful IF verification environment [7] by preserving the original structure as much as needed. On the other hand, we paid attention to defining a translation which is as flexible as possible, due to the fact, that UML has many possible semantic interpretations, so that it would be easy to adapt to different profiles, where we target in particular profiles which would allow both asynchronous and synchronous signals and/or operation calls, and different execution modes. This is obtained by using (asynchronous) signals as a basic communication means, and adding a layer of dynamic priority rules (in the spirit of the approach proposed in [11, 12]) in order to more or less restrict the possible execution orders.

Within OMEGA, [13] studied a general syntactic and semantic framework of a real-time profile for UML, which is compatible with the proposals of the OMG's UML profile for scheduling, performance and time. This profile is based on a very general and expressive notion of event (instant of state change), which allows to express timing assumptions and requirements as independently as possible from the operational model. One originality of the approach presented here, is that it provides a formal semantics of the time related

primitives in terms of timed automata with deadlines. An interesting point is that this time extension is independent of the dynamic semantics of the functional part. It is this timing profile that has been implemented in the above mentioned tool. As it is, only parts of this profile are likely to be used in practise, such as the notion of timed observer that has been introduced as a stereotype of state machines, and a number of patterns defined on the top of the general language, expressing particular durations (response times, transmission delays, execution times,...). The advantage here is that the meaning of such patterns can be defined precisely by identifying the events defining these durations, whereas in UML they are just keyword, the precise meaning is left to tools. In OMEGA, activity diagrams have not be considered (to keep focus). Nevertheless, they are useful in the context of scheduling, and for this reason we did study them in AMETIST, see below. Events as defined in the profile can be used in a future version to unambiguously define the relationship between the state machine and the activity diagram view.

AMETIST end-user panel member ASML uses UML activity diagrams to specify scheduling problems that it encounters in the area of manufacturing systems. This notation is particularly useful to give compact specifications of repetitive behavior using counters and conditionals. As part of the AMETIST project, we identified a subclass of UML activity

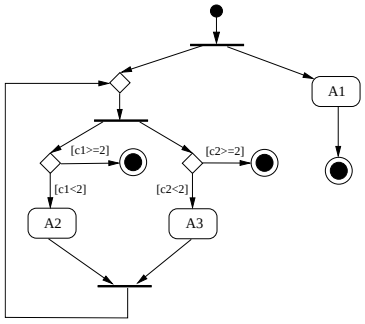

Figure 1: A small activity graph.

diagrams which can be translated/expanded to task graphs [18] (see the examples in Figure 1 and Figure 2). In the specification of production processes we typically see nested finite repetitions, which blows up the size of the underlying task graph and, in an even worse way, the number of possible schedules. For instance, a manufacturing order of a beer brewery consists of several pallets, containing several crates with several bottles of beer. Another example concerns a wafer scanner manufacturing system from the semiconductor industry. Wafers are also produced in batches (lots). A wafer scanner projects a mask on a wafer, using light. Eventually, the projected masks result in Integrated Circuits (ICs). On one wafer, multiple ICs and types of ICs are manufactured. Multiple types of ICs involve multiple masks, and multiple masks are placed on a reticle. We showed how schedules for instances of sequence diagrams with small bounds on counters, can be used to build schedules for diagrams with large bounds on these counters, thus effectively fighting state space explosion. It is future work to study conditions which ensure preservation of optimality of schedules after change of maximal counter values.

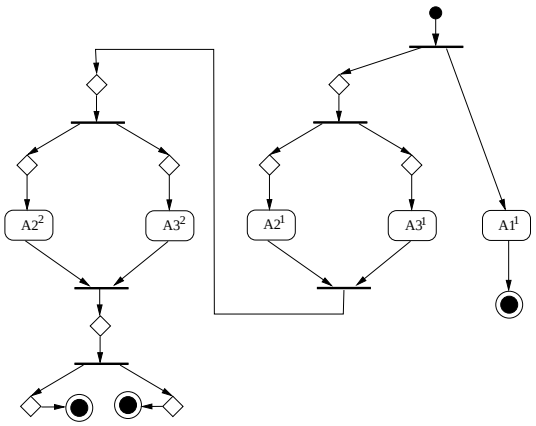

Figure 2: The intended unfolding of the activity graph of Figure 1.

4.1 Conclusions

Although UML has not been designed originally for embedded systems, one can observe an increasing use of object-oriented techniques and UML in this domain. There exists a UML profile for Schedulability, Performance and Time, several dedicated CASE tools (e.g., Artisan's Real-time Studio, Rhapsody of I-Logix, Rational Rose RealTime, and Telelogic TAU). We therefore believe that establishing strong links between notation+tools for UML, and notation+tools for timed automata may greatly help to increase the use/usability of the AMETIST methods.

The OMEGA project has successfully established such translations, and has used, for instance, the MARS case study of NLR to experiment with various forms of formal support for UML-based development, including requirements capturing by LSCs, and timed and non-timed model checking. Clearly, scalability is an important problem for all techniques. In the MARS example, small models lead to state space explosion problems for model checkers, e.g., due to the asynchronous data sources. Moreover, the large number of features in UML (such as synchronous and asynchronous communication, threads of control, and hierarchical state machines) creates an additional layer of complexity, since the semantics of all these features has to be included in the tools [19].

We expect that UML notations will also increasingly be used to study resource allocation problems of the type studied within AMETIST, see for instance [18].

Much more research will be required to make model checking of TA models generated from industrial UML specifications practical. On the one hand this will involve the application of general abstraction techniques (such as counterexample guided abstraction refinement), on the other hand it will require the exploitation of the model structure induced by UML specifications.

5 Scenario Based Specification

Sequence Charts (whether the original MSCs or their UML variants, sequence diagrams) are quite popular amongst software designers, e.g., to describe the typical behaviors of some communication protocol at an early stage of its design. The important question of whether some given set of scenarios is realizable by some message passing system has already been investigated in different ways. In [2], we consider deadlock-free implementations up to some refinement of message contents. We present an algorithm to check safe implementability of MSC specifications in the framework of non-FIFO channels. Our criterion turns out to have the same complexity bounds as the restrictive approach of Alur et al. where no additional message content is allowed. The construction algorithm proposed refines a given protocol so that to get a safe implementation that adds control information to messages in order to forbid any overflow of the channel capacities. The result has applications also in the area of synthesis of distributed controllers from distributed specifications.

Sequence charts possess a rather weak partial order semantics that does not make it possible to capture many kinds of behavioral requirements of a system. This is mostly due to the fact that they are not able to distinguish between possible and mandatory behavior. To address this, while remaining within the general spirit of scenario-based visual formalisms, a broad extension was proposed in 1999, termed *live sequence charts* (LSCs). Among many other things, LSCs can express forbidden behavior ("anti-scenarios"). The expressiveness of the extended language makes it possible to view LSCs as a rather powerful executable model, and not only as a transient development product to be used for verification and documentation. Indeed, in [14, 15, 16] a *play-out* execution mechanism is described. Using play-out, the user can execute requirement specifications given in the inter-object style of LSCs directly, without the need to build or synthesize a system model consisting of state machines or code. The play-out mechanism is one part of a wider methodology called *play in/out*, also presented in [14, 15, 16]; the other part enables scenarios to be played in directly from a GUI or an object model diagram. An extreme and tantalizing possibility would involve systems for which the LSC specification, together with the play engine, can be taken to be not just the requirements but the final implementation.

From the perspective of AMETIST it is interesting to see how timing constraints can be specified in LSCs, see [15]. In [30, 25, 17], it is shown that LSCs provide natural modeling language for systems such as the smart-card personalization machine of partner Cybernetix. LSCs is taken as an example of a scenario based specification language which enables better modeling procedure using the Play-In/Play-Out approach that is implemented in the prototype tool PlayEngine. It has also been demonstrated that it is possible to synthesize a scheduler with Smart Play-Out via a translation to the (untimed) model checker SMV.

5.1 Conclusions

The research on scenario based specification has also been common work with OMEGA. Within OMEGA, the focus was on validation of the consistency of a set of LSCs, on vali-

dation of properties expressed by a set of LSCs on an operational UML specification, and on the generation of state machines from LSCs. Within AMETIST, the focus was on experimenting with the play-in/play-out approach on an industrial example. The interaction with the timed automata approach has turned out to be quite interesting.

In UPPAAL, message sequence charts are used at the output side, to visualize counterexamples and schedules computed by the model checker. In the PlayEngine, the related LSCs are used on the input side, as the basic elements from which models are composed.

A central idea in the play-in/play out approach is that a graphic user interface of the system is constructed and then used to specify the requirements in the play-in stage and to show the execution during play-out. There are certain applications for which such a graphical representation is particularly natural and straightforward, for instance the smartcard personalization system of Cybernetix. Certainly, the convenience of using UPPAAL could be further enhanced by adding a similar GUI feature for visualizing traces, in addition to the existing simulator and Gantt chart generator.

There are several interesting possibilities for for tool interaction. In particular a translation of LSCs to TA makes sense. Such a translation may serve as a tool to analyze timing constraints and synthesize schedulers within the framework of the play-in play-out methodology.

References

[1] T. Arons, J. Hooman, H. Kugler, A. Pnueli, and M. van der Zwaag. Deductive verification of UML models in TLPVS. In *Proceedings UML 2004*, pages 335–349. LNCS 3273, Springer-Verlag, 2004. Available from World Wide Web: `http://www.cs.ru.nl/~hooman/UML2004.html`.

[2] N. Baudru and R. Morin. Safe implementability of regular message sequence chart specifications. In *Proceedings of the ACIS Fourth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'03)*, 2003.

[3] S. Bornot, G. Gößler, and J. Sifakis. On the construction of live timed systems. In Susanne Graf and Michael I. Schwartzbach, editors, *6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *Lecture Notes in Computer Science*, pages 172–202. Springer-Verlag, April 2000.

[4] Sébastien Bornot and Joseph Sifakis. An algebraic framework for urgency. *Inf. Comput.*, 163(1):172–202, 2000.

[5] Howard Bowman. Modelling timeouts without timelocks. In *ARTS'99, 5th International AMAST Workshop on Real-time and Probabilistic Systems*, LNCS, page 20. Springer-Verlag, May 1999.

[6] M. Bozga, S. Graf, and L. Mounier. If-2.0: A validation environment for component-based real-time systems. In K.G. Larsen Ed Brinksma, editor, *Proceedings of CAV'02*, volume 2404 of *LNCS*, pages 343–348, Copenhagen, Denmark, July 2002. Springer. Available from World Wide Web: `http://www-verimag.imag.fr/~async/BIBLIO/papers/Bozga-Graf-Mounier-02.ps.gz`.

[7] M. Bozga, S. Graf, and L. Mounier. If-2.0: A validation environment for component-based real-time systems. In K.G. Larsen Ed Brinksma, editor, *Proceedings of CAV'02 (Copenhagen, Denmark)*, volume 2404 of *LNCS*, pages 343–348. Springer-Verlag, July 2002. Available from World Wide Web: `http://www-verimag.imag.fr/~graf/biblio-abstr.html#CAV02`.

[8] M. Bozga and O. Maler. Timed automata approach for the axxom case study. Technical report, Verimag, 2003. Available from World Wide Web: `http://www-verimag.imag.fr/~maler/AMETIST/axxom-report.pdf`.

[9] Pedro R. D'Argenio. A compositional translation of stochastic automata into timed automata. Technical report, University of Twente, 2000. CTIT-00-08.

[10] Biniam Gebremichael and Frits Vaandrager. Specifying urgency in timed I/O automata. In B.K. Aichernig and B. Beckert, editors, *3rd IEEE International Conference on Software Engineering and Formal Methods (SEFM05)*, pages 64–73, Koblenz, Germany, September 2005. IEEE Computer Society. Available from World Wide Web: `http://www.cs.ru.nl/ita/publications/papers/fvaan/urgency.html`.

[11] Gregor Gößler and Joseph Sifakis. Composition for component-based modeling. In *proceedings of FMCO 2002, Leiden, the Netherlands*, LNCS 2852, pages 443–466, 2002. Available from World Wide Web: `http://www-verimag.imag.fr/~sifakis/`.

[12] Gregor Gößler and Joseph Sifakis. Component-based construction of deadlock-free systems. In *proceedings of FSTTCS 2003, Mumbai, India*, LNCS 2914, pages 420–433, December 2003. Available from World Wide Web: `http://www-verimag.imag.fr/~sifakis/`.

[13] Susanne Graf, Ileana Ober, and Iulian Ober. A real-time profile for uml. *STTT*, 8(2):113–127, 2006. Available from World Wide Web: `http://dx.doi.org/10.1007/s10009-005-0213-x`.

[14] D. Harel, H. Kugler, R. Marelly, and A. Pnueli. Smart play-out of behavioral requirements. In *Proc. 4^{th} Intl. Conference on Formal Methods in Computer-Aided Design (FMCAD'02), Portland, Oregon*, volume 2517 of *Lect. Notes in Comp. Sci.*, pages 378–398, 2002. Available from World Wide Web: `http://www.wisdom.weizmann.ac.il/~dharel/papers/FMCAD02.pdf`. Also available as Tech. Report MCS02-08, The Weizmann Institute of Science.

[15] D. Harel and R. Marelly. Playing with time: On the specification and execution of time-enriched LSCs. In *Proc. 10th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'02)*, Fort Worth, Texas, 2002. Available from World Wide Web: `http://www.wisdom.weizmann.ac.il/~dharel/papers/TimedLSCs.pdf`.

[16] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, 2003. Available from World Wide Web: `http://www.wisdom.weizmann.ac.il/~dharel/comeplay.html`.

[17] David Harel, Hillel Kugler, and Gera Weiss. Some methodological observations resulting from experience using lscs and the play-in/play-out approach. In *Proc. Scenarios: Models, Algorithms and Tools*, volume 3466 of *Lect. Notes in Comp. Sci.*, pages 26–42. Springer-Verlag, 2005. Available from World Wide Web: `http://www.wisdom.weizmann.ac.il/~gera/methodology.pdf`.

[18] M. Hendriks, N.J.M. van den Nieuwelaar, and F.W. Vaandrager. Recognizing finite repetitive scheduling patterns in manufacturing systems. In G. Kendall, E. Burke, and S. Petrovic, editors, *Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2003),* Nottingham, UK, *Volume I*, pages 291–319. The University of Nottingham, August 2003. Available from World Wide Web: `http://www.cs.kun.nl/ita/publications/papers/fvaan/HNV03.html`. ISBN 0-9545821-0-1.

[19] J. Hooman, H. Kugler, I. Ober, A. Votintseva, and Y. Yushtein. Supporting UML-based development of embedded systems by formal techniques. *International Journal of Software and Systems Modeling (SoSym)*, 2007. Available from World Wide Web: `http://www.cs.nyu.edu/~kugler/NLR07.pdf`. To appear.

[20] J. Hooman and M. van der Zwaag. A semantics of communicating reactive objects with timing. *Int. Journal on Software Tools for Technology Transfer (STTT)*, 8(4):97–112, 2006. Available from World Wide Web: `http://www.cs.ru.nl/~hooman/STTTpvs.html`.

[21] J. Hooman and M.B. van der Zwaag. A semantics of communicating reactive objects with timing. In *Proceedings SVERTS Workshop of the Sixth International Conference on the Unified Modeling Language, UML 2003*, 2003. Available from World Wide Web: `http://www-verimag.imag.fr/EVENTS/2003/SVERTS/PAPERS-WEB/13-HoomanZwaag.pdf`.

[22] D.K. Kaynar, N.A. Lynch, R. Segala, and F.W. Vaandrager. A framework for modelling timed systems with restricted hybrid automata. In *Proceedings of the 24th International IEEE Real-Time Systems Symposium (RTSS03),* December 3-5, 2003, Cancun, Mexico, pages 166–177. IEEE Computer Society, 2003. Available from World Wide Web: `http://www.cs.kun.nl/ita/publications/papers/fvaan/RTSS03.html`.

[23] D.K. Kaynar, N.A. Lynch, R. Segala, and F.W. Vaandrager. The theory of timed I/O automata. Technical Report MIT-LCS-TR-917, MIT Laboratory for Computer Science, Cambridge, MA, 2003. Available from World Wide Web: `http://theory.lcs.mit.edu/tds/papers/Kirli/TIOA-TR.ps`.

[24] D.K. Kaynar, N.A. Lynch, R. Segala, and F.W. Vaandrager. *The Theory of Timed I/O Automata*. Morgan & Claypool Publishers, 2006. Available from World Wide Web: `http://www.ita.cs.ru.nl/publications/papers/fvaan/synthesis.html`. Synthesis Lecture on Computer Science, 101pp, ISBN 159829010X.

[25] Hillel Kugler and Gera Weiss. Planning a production line with LSCs. Research report, Weizmann, 2004. Available from World Wide Web: `http://ametist.cs.utwente.nl/INTERNAL/PUBLICATIONS/WISPublications/cybernetix.zip`.

[26] M. Kyas and J. Hooman. Compositional verification of timed components using pvs. In B. Biel, M. Book, and V. Gruhn, editors, *Proceedings of Software Engineering 2006*, volume P-79 of *Lecture Notes in Informatics*, pages 143 – 154, 2006. Available from World Wide Web: `http://www.cs.ru.nl/~hooman/SE06.html`.

[27] Iulian Ober, Susanne Graf, and Ileana Ober. Model checking of UML models via a mapping to communicating extended timed automata. In *SPIN'04 Workshop on Model Checking of Software, 2004*, volume LNCS 2989, pages 127–145, 2004. Available from World Wide Web: `http://springerlink.metapress.com/content/2u0c0n4a5r10kkqd/fulltext.pdf`.

[28] J. Sifakis and S. Yovine. Compositional specification of timed systems (extended abstract). In Claude Puech and Rüdiger Reischuk, editors, *STACS*, volume 1046 of *Lecture Notes in Computer Science*, pages 347–359. Springer, 1996.

[29] F.W. Vaandrager and A.L. de Groot. Analysis of a biphase mark protocol with Uppaal and PVS. Technical Report NIII-R0445, NIII, Radboud University Nijmegen, 2004. Available from World Wide Web: `http://www.cs.kun.nl/ita/publications/papers/fvaan/BMP.html`. Published in Formal Aspects of Computing Journal 18(4):433-458, December 2006.

[30] Gera Weiss. Modeling smart-card personalization machine with LSCs. Research report, Weizmann, 2003. Available from World Wide Web: `http://ametist.cs.utwente.nl/INTERNAL/PUBLICATIONS/WISPublications/cybernetix.zip`.