Modular Verification of Safe Online-Reconfiguration for Proactive Components in Mechatronic UML*

Holger Giese and Martin Hirsch**

Software Engineering Group, University of Paderborn,
Warburger Str. 100, D-33098 Paderborn, Germany
{hg, mahirsch}@uni-paderborn.de

Abstract. While traditionally the environment considered by an autonomous mechatronic systems only consists of the measurable, surrounding physical world, today advanced mechatronic systems also include the context established by the information technology. This trend makes mechatronic systems possible which consist of cooperating agents which optimize and reconfigure the system behavior by adjusting their local behavior and cooperation structure to better serve their current goals depending on the experienced mechanical and information environment. The MECHATRONIC UML approach enables the component-wise development of such self-optimizing mechatronic systems by providing a notion for hybrid components and support for modular verification of the safe online-reconfiguration. In this paper, we present an extension to the formerly presented solution which overcomes the restriction that only purely reactive behavior with restricted time constraints can be verified. We present how model checking can be employed to also verify the safe modular reconfiguration for systems which include components with complex time constraints and proactive behavior.

1 Introduction

To realize advanced mechatronic systems such as intelligent cooperating vehicles, the engineers from the different disciplines mechanical engineering, electrical engineering, and software engineering have to cooperate successfully. The development of such systems becomes even more challenging, if the mechatronic systems should be able to adjust their behavior and structure at run-time (cf. self-adaptation and self-optimization [1]).

The environment of autonomous mechatronic subsystems today no longer consists only of the physical world. In addition, the context built by the interconnection of the system via information technology such as local bus systems or wireless networking technology has to be taken into account. Therefore, today more flexible mechatronic systems are developed which require complex online reconfiguration schemes for the control algorithms which can effect not only a single component but whole hierarchies of connected components.

* This work was developed in the course of the Special Research Initiative 614 - Self-optimizing Concepts and Structures in Mechanical Engineering - University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

** Supported by the University of Paderborn.

J.-M. Bruel (Ed.): MoDELS 2005 Workshops, LNCS 3844, pp. 67–78, 2006.

The MECHATRONIC UML approach provides an approach for the component-wise development of such complex self-optimizing mechatronic systems. It supports a component notion and the specification of required online reconfiguration activities which go across multiple levels of a component hierarchy [2]. Therefore, online-reconfiguration can be employed also on the higher levels of the system control but also across the boundaries traditionally given by the involved disciplines. The MECHATRONIC UML approach also supports model checking techniques for real-time processing at the higher levels. It addresses the scalability problem of these techniques by supporting a compositional proceeding for modeling and compositional verification of the real-time software when using the UML 2.0 component model and the corresponding definition of ports and connectors as well as patterns [3].

In [2], an approach to combine such a compositional approach with techniques to ensure the proper modular reconfiguration has been presented which require a rather restricted purely reactive behavior and rather restricted local timing constraints for the subordinated components. More complex timing constraints including clock invariants which enforce certain reconfiguration sequences or proactive behavior in the sense that the subordinated component autonomously decides that a reconfiguration is required is currently not supported. In this paper, we present how the verification of the safe reconfiguration can be accomplished with model checking to also cover these two cases.

The paper proceeds with an informal introduction on modeling with the MECHATRONIC UML approach by means of an example given in Section 2 and some remarks on the current tool support. To point to the limitations of the existing approach the example is extended with proactive behavior and non complex timing constraints which effect the reconfiguration. The concepts and first evaluation results for verifying the safe reconfiguration follow in Section 4. We close the paper with a discussion of related work (Section 5) and a final conclusion including an outlook on planned future work in Section 6.

2 Modeling and Current Tool Support

In this section, we introduce our MECHATRONIC UML approach focusing on modeling with hybrid components (cf. [2]). Furthermore, we present our current tool support for modeling and verification of hybrid systems.

To outline our approach, we employ an example which stems from the RailCab[1] research project at the University of Paderborn. Autonomous shuttles are developed which operate individually and make independent and decentralized operational decisions.

The shuttle's active suspension system and its optimization is one example for a complex mechatronic system whose control software we design in the following. The schema of the relevant physical model of our example is shown in Figure 1(a).

The suspension module is based on air springs which are damped actively by the displacement of their bases and three vertical hydraulic cylinders which move the bases of the air springs via an intermediate frame – the suspension frame. The vital task of the system is to provide the passengers a high comfort and to guarantee safety when

[1] http://www-nbp.upb.de/en/index.html

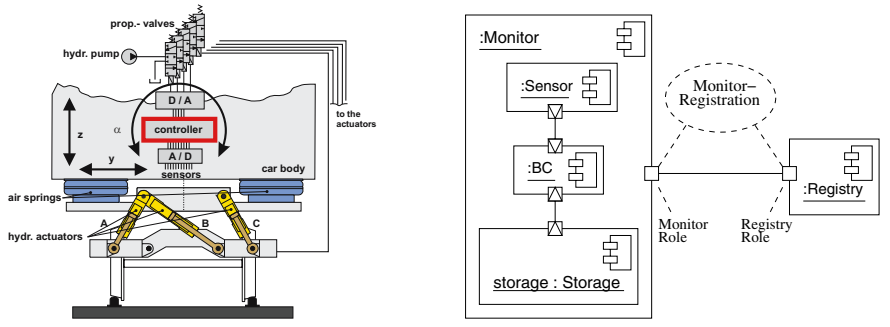

Fig. 1. 1(a) Scheme of the suspension module / 1(b) Monitor and its environment

controlling the shuttle's car body. In order to achieve this goal, multiple feedback controllers are used with different capabilities in matters of safety and comfort [4].

We focus on 3 controllers which provide the shuttle different comfort: The Reference controller provides sophisticated comfort by referring to a trajectory describing the required motion of the coach body in order to compensate the current track's unevenness, slopes, etc. To guarantee stability, all sensors have to deliver correct values. In case of e.g. incorrect values the less comfortable Absolute controller has to be applied, which requires only the vertical acceleration as input. If this sensor fails, our Robust controller, which provides the lowest comfort, but requires just standard inputs to guarantee stability, has to be applied. We have to distinguish between two different cases: *atomic switching* and *cross fading*. In the case of atomic switching, the change takes place between two computation steps. If the operating points of the controllers are not identical, it will be necessary to cross-fade between the two controllers.

The architecture of the suspension module is depicted in Figure 1(b). The Monitor component coordinates its embedded components BC, Sensor, and Storage. Further, it communicates via the MonitorRegistration pattern with the Registry. If Registry sends the information about the upcoming track section to Monitor, the Monitor stores it in the Storage component. Sensor provides the signals. To model the hierarchical embedding of the BC component into the Monitor component, aggregation for UML 2.0 components is used. The non-hierarchical link of the Monitor component to the Registry component is described by two ports (as defined in the UML 2.0 as unfilled boxes) and a connector.

To additionally model the quasi-continuous aspects of the model in form of communication via continuous signals, we extend the UML by *continuous ports*, depicted by framed triangles whose orientation indicates the direction of the signal flow. The behavior of the hybrid component is specified by means of an extension of UML State Machines called *hybrid reconfiguration charts*. We employ Real-Time Statecharts [5] to describe required real-time behavior and we describe the continuous behavior by embedding appropriate basic quasi-continuous block configurations (cf. the BC component behavior in Figure 2(a)).

While a common hybrid automaton specification requires always the same input and output signals for every location, the required controller logic with its specific required input and provided output signals is specified within each state of a hybrid reconfig-

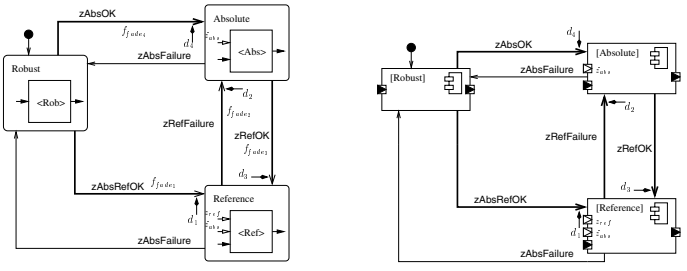

Fig. 2. Behavior description of the BodyControl (2(a)) and the interface state chart (2(b))

uration chart (cf. Figure 2(a)). The continuous ports that are required in each of the three interfaces are filled black, the ones that are only used in a subset of the states are filled white. In our notion of hybrid reconfiguration charts, we introduce additional *fading-transitions*, which are visualized by thick arrows, while atomic switches have the shape of regular arrows. Parameters of a transition are: A source- and a target-location, a guard and an event trigger, information on whether or not it is an atomic switch, and, in the latter case, a fading strategy (f_{fade}) and the required fading duration interval $d = [d_{low},\ d_{up}]$ specifying the minimum and maximum duration of fading.

For embedding or connecting a hybrid component, we do not need all details of the component realization, but only enough information about its externally observable behavior such that compatibility can be analyzed. In Figure 2(b) the related *interface state chart* of the BC component is displayed. The interface state chart abstracts from the continuous behavior, it still contains the information about the input-output dependencies and permits us to abstract from all internal variables and internal signals.

Therefor, we present in the following a concept for the behavioral embedding of the subcomponents within the hybrid reconfiguration charts of a component, which permits to check consistency w.r.t. reconfiguration at a purely syntactical level.

The behavioral embedding of subcomponents is realized by assigning a configuration of aggregated subcomponents (not only quasi-continuous blocks) to each state of a hybrid reconfiguration chart by means of UML instance diagrams (see Figure 3). A switch between the states of the monitor chart then *implies* a switch between states of the interface state charts of the embedded components.

The behavior of the Monitor component is specified by a hybrid reconfiguration chart (cf. Figure 3). We have assigned the BC component in the appropriate state to each state of the upper orthogonal state of the chart. E.g., the BC component instance in state Reference has been (via a visual embedding) assigned to the location AllAvailable of the monitor where sensors z_{ref} as well as $\ddot{z}_{abs}$ are available. The communication with the Registry is described in the lower orthogonal state of Figure 3 (cf. [2]). The upper orthogonal state consists of the states RefAvailable and AllAvailable which represents whether the required reference curve is available for the *current* track. The upper state is synchronized by the lower one.

For the outlined MECHATRONIC UML approach there exists tool support. Especially two specific verification tasks for the resulting systems are supported. At first the real-time coordination of the distributed software, which is modeled with UML 2.0

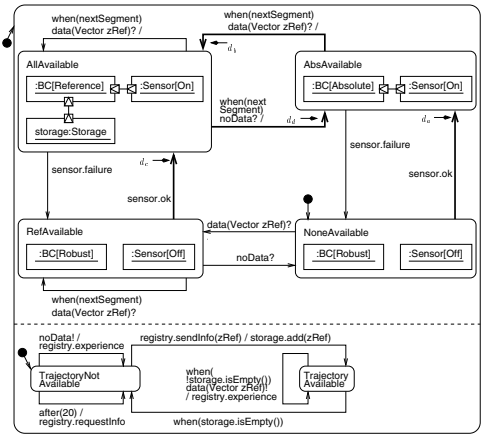

Fig. 3. Behavioral embedding in the Monitor realization

components and connectors is verified using a compositional model checking approach based on verified coordination patterns [3]. Secondly, a restricted subset of the outlined hierarchical component structures for modeling of discrete and continuous control behavior is checked for the consistent reconfiguration and proper real-time synchronization w.r.t reconfiguration [2]. In addition, the second approach can be embedded into the first one.

The current results enable the systematic development of complex mechatronic systems with safe online-reconfiguration, as in the current practice of mechatronic systems the strict hierarchical approach with strict top-down command order is standard. However, a number of limitations result which seem to unduly restrict the design space for more advanced modular designs of mechatronic systems in the future.

3 Complex and Proactive Behavior

One severe restriction is that in interface state charts only the duration of transitions can be specified but not their time-triggered execution. Examples where an extension to more general time constraints in the interface state chart seem beneficial are, for example, restrictions on the frequency of mode switches. In our example, the engineer could more easily ensure the stability of the underlying system if the interface state chart restricted that after a switch from state Reference to state Absolute a certain time threshold should elapse before the BC component permits to switch back to the more comfortable Reference state.

Another limitation is the strict top-down command order. While the processing of the sensor error already indicates that we have to consider errors of the subcomponents, the current form of interface state charts does not permit to encode a required reaction time or switching of the mode within the interface. Thus, whether a required reaction of the embedding component results or not is currently not included in the interface. Therefore, besides only emitting warnings, error reports, or wishes to the embedding

component, true *proactive* behavior in the interface state charts seems favorable such that when sending an error report the interface state chart can also initiate that within a given time frame the current state has to be left. As example for proactive behavior, consider the case that BC component detects that the operation with the reference data results in unexpected problems and wants to report this to the embedding Monitor component, the BC component may in addition specify in the interface state chart that therefore the reference mode has to be left within a given deadline to ensure that the observed behavior is not critical.

We can characterize these cases where more expressive notion of interface automaton are required as follows: An interface automaton M is *complex* if it is not simple but still deterministic. An interface automaton M is *proactive* if it autonomously decides that a reconfiguration is required which results in a non-deterministic behavior.

In Section 2, the suspension module was introduced and thereon the control software was modeled. One characteristic of the control software was the top-down command order. It was not possible for the BC component to influence the Monitor component via direct events. E.g. if any error occurs when the BC component is in the Reference state, the BC component has to switch to the Robust state and has to inform the superordinate Monitor component to react in an adequate way. Furthermore, we want to avoid perpetual uncontrolled switching between Absolute and Reference.

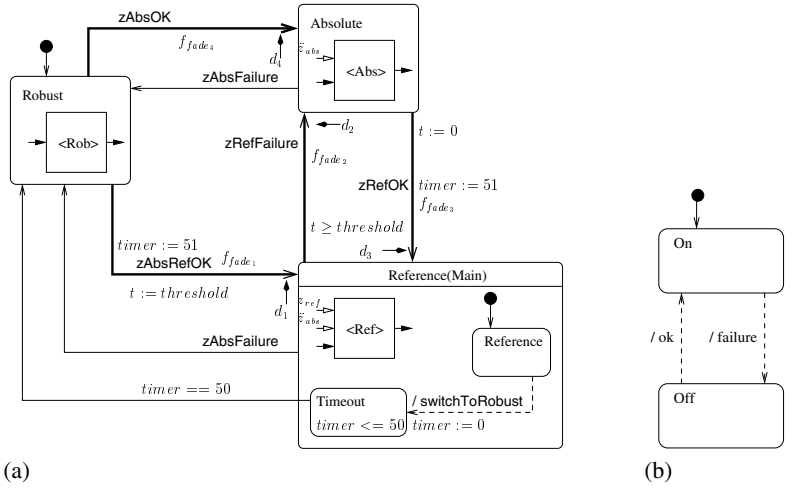

Fig. 4. Behavior description of the BC component (4(a)) and Sensor component (4(b))

In Figure 4(a), the redesigned BC component is depicted. The behavior of the former BC component is extended by proactive behavior. When the BC component is in the Reference state, the component is now able to decide autonomously to switch to the Robust state. Due to non-urgent transitions (dashed line), non-determinism is introduced. This is modeled as follows: While staying in the Reference state, the body control sends a message switchToRobust to the superordinate Monitor component. This done, the BC component pauses in a Timeout state. If the timeout is reached, the BC

component switches to Robust. To control the switching between Absolute and Reference, a timer t is introduced. Everytime the Reference state is entered from the Absolute state, a clock t is set to zero. To avoid an immediate return back to Absolute, a guard, representing a threshold, t>=threshold, is added to the transition. All other incoming transitions to the Reference state get an additional assignment t:=threshold, thus the threshold is "omitted". In Figure 4(b) the interface state chart of the sensor component, consisting of two states, on and off, is depicted.

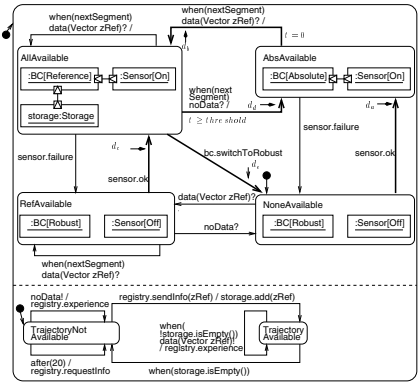

Fig. 5. Behavioral embedding in the Monitor realization

Similar to Figure 3, the behavior of the Monitor component is depicted in Figure 5. In addition to the old Monitor component, we have to take into account the proactive and timing behavior of the subordinated components. Since the body control now sends the switchToRobust message, the monitor component has to consume this message.

4 Checking Complex and Proactive Subcomponents

To adjust our modular reasoning approach to the extensions outlined in the last section, we have to provide checks for refinement and correct embedding which support the introduced more expressive interface automata.

4.1 Checking Refinement

Complex and proactive components do not result in a *simple* interface state chart and thus the checking procedure for ensuring that the interface state chart corresponds to the component behavior is not applicable.

In [6], an approach for checking the employed notion of refinement ($M \sqsubseteq_{RT} M^I$) has been presented which requires that M^I is deterministic. If the interface state chart M^I is complex but not proactive, we can thus employ this approach. For a deterministic M^I we have to derive a corresponding test automaton M_t^I as described in [6] and then check $M \| M_t^I$ for time stopping deadlocks.

If, however, the interface state chart M^I is proactive and thus not deterministic, we have to look for alternatives, as we cannot derive a deterministic timed automaton for each non-deterministic one (cf. [7]). Analyzing the limitation of the approach outlined in [6], we can conclude that the branching within the on-the-fly traversed cross-product simply requires that a unique mapping to a state in the refined model exists which is guaranteed in [6] by the deterministic character of M^I.

We propose to exploit the mapping $map : L_p^I \rightarrow L$ between the passive states of the interface automaton and related states in the underlying realization to achieve a feasible solution for this case. For a mapping map which assigns to each realization state exactly one state of the interface automaton and thus map^{-1} is a function and we write $l' = map^{-1}(l))$ and the case that no two transitions with the same source location, label, and target location exist, we can build syntactically the cross-product $M' = M^I \times_{map} M$.

We can then simply check whether a time stopping deadlock or a *bad state* can be reached in M' and conclude that refinement holds if no such violation has been detected. A more detailed and technical description of the mapping can be found in [8] and will be omitted here.

To verify if the hybrid reconfiguration automaton of the BC component is a correct refinement of the BC component interface state chart, we have to built a timed automata model as explained above. We use the model checker UPPAAL [9] for the verification and check the constraints, formulated in TCTL, A[] not deadlock and E<> BodyControl.Error ensuring the correct refinement. The verification took about 1.21 seconds and at maximum about 8032 KB were allocated by the verifier[2].

4.2 Checking the Embedding

To realize the dynamic checking for the prove of correct reconfiguration behavior, the hybrid reconfiguration chart and the interface state chart have to be transformed to an appropriate model which can be handled by a model checker. In [10], transformation rules from Real-Time Statecharts to timed automata were introduced. In the following, we reuse and extend theses transformation rules.

In hybrid reconfiguration charts, component instances are embedded in locations. During the transformation, the instances are omitted because the associated interface state charts are also transformed. Due to this fact, for the transformation of locations, we can apply the same ones as for Real-Time Statecharts.

In addition to the locations, the transitions have to be transformed. In contrast to Real-Time Statecharts, a transition is associated with a fading function. Since the fading function does not affect the real-time behavior, it is omitted, too. Hence the same transformation as for Real-Time Statecharts is used. As mentioned before, the interface state charts of the embedded component instances have to be transformed. It has to be guaranteed that the embedded component instance leaves an internal location iff the superordinate component leaves a location. For example, when the monitor component leaves the location NoneAvailable, the embedded component instance BC has to leave the internal location Robust. To achieve this behavior, we use the synchronization semantics from the timed automata model. The superordinate component has the

[2] The verification was done on a Pentium 4, 2.4 GHz, 1 GB memory, OS Linux Redhat.

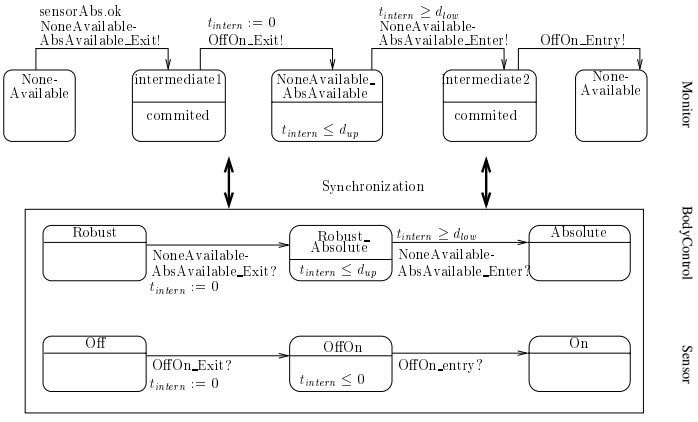

Fig. 6. Synchronization between Monitor, Sensor, and BodyControl

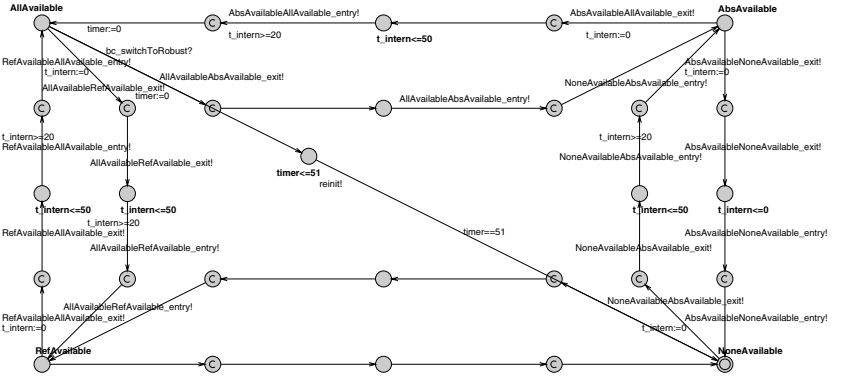

Fig. 7. Timed Automaton of the Monitor state chart

meaning of a sender (!), whereas the embedded component instances have the meaning of a receiver (?). In the case of more than one embedded component instance, we use a chain of committed locations (cf. [9]) for synchronization. In Figure 6, an example is depicted.

For the evaluation, we use the real-time model checker UPPAAL [9]. As an example the transformed timed automata of the interface state chart of the of the Monitor component is depicted in Figure 7. For clarity, we do not depict the timed automaton of the Sensor (the transformation is rather trivial). For the verification the automata are executed in parallel. We check the property A[] not deadlock. As result of the verification, we get that the system is deadlock free. In particular, this means we have a correct embedding of all components. The verification took about 0.31 seconds and at maximum about 2092 KB were allocated by the verifier.

5 Related Work

The *de facto* industry standard for modeling of mechatronic systems with hybrid behavior is MATLAB/Simulink and Stateflow[3]. Formal verification of MATLAB/Simulink and Stateflow models of moderate size can be accomplished by automatically transforming them to hybrid automata (cf. [11]).

Besides MATLAB/Simulink and Stateflow, there are also a number of approaches, like Charon [12], Masaccio [13], HyCharts & HyRoom [14], and Hybrid I/O Automata [15], which address the problem of modeling complex systems by some form of hybrid state charts. Some of them support hierarchy and parallelism as well as a notion of component and some of them even support formal verification.

All existing approaches fail in providing a component concept which supports a dynamic interface which enables to decompose systems with online reconfiguration into multiple hybrid state charts. Thus, a usually not feasible check of the whole system is required to ensure that a system with complex reconfiguration such as the presented example is correct w.r.t. the reconfiguration and real-time behavior.

Another consequence is that in these approaches the control engineering know-how for the continuous control and the software engineering know-how for the real-time coordination have to be specified both within a single hybrid component and thus a tight cooperation between engineers from different camps is required. In our approach, in contrast, an interface between the control engineering specific details and the more software engineering oriented distributed real-time processing is possible which support more realistic loosely coupled development processes.

Available compositional reasoning approaches for hybrid systems [16, 17, 18] require high manual effort of inventing auxiliary properties to enable a full verification to decide whether the described reconfiguration is consistent. The presented approach in contrast will ensure consistency by means of a syntactical check or modular model checking of the separate components and their interfaces.

We employ in our approach the algorithm for checking the refinement relation between timed automata as proposed in [6]. As outlined in the paper, for proactive and thus non-deterministic interface automata this approach is not applicable.

In [19], an algorithm for checking the existence of a simulation relation to investigate the opportunities of refinement checking for Cottbus Timed Automata is developed. The approach is restricted to simulation and closed timed automata with integer semantics, while we require a stronger form of refinement (ready simulation).

6 Conclusion and Future Work

Within this paper, we presented an incremental improvement of our modular verification approach for checking that the online-reconfiguration of MECHATRONIC UML models is safe. In our earlier proposal [2], severe restrictions to the expressiveness of the supported component interface are employed to ensure that efficient checks can be used which do not have to consider the whole state space. In this paper, we present support for more expressive interfaces which include complex timing constraints and

[3] http://www.mathworks.com

proactive behavior employing model checking. The underlying concepts are outlined and formally defined. In addition, first experimental results are reported.

While modeling of hybrid systems with the MECHATRONIC UML approach is already supported by the FUJABA Real-Time Tool Suite[4], the described refinement checks and the check for the correct embedding are currently under development.

In future work, we plan to further improve and extend our approach w.r.t. modeling and verification such that the full hybrid behavior of the components is covered. On particular next step is to apply the general checking procedure for simulation as presented in [19] for checking our notion of refinement by generalizing our extension of the automata presented in Section 4.1 (cf. [8]).

References

1. Janos Sztipanovits, Gabor Karsai, and Ted Bapty. Self-adaptive software for signal processing. *Commun. ACM*, 41(5):66–73, 1998.
2. Holger Giese, Sven Burmester, Wilhelm Schäfer, and Oliver Oberschelp. Modular Design and Verification of Component-Based Mechatronic Systems with Online-Reconfiguration. In *Proc. of 12th ACM SIGSOFT Foundations of Software Engineering 2004 (FSE 2004), Newport Beach, USA*, pages 179–188. ACM Press, November 2004.
3. Holger Giese, Matthias Tichy, Sven Burmester, Wilhelm Schäfer, and Stephan Flake. Towards the Compositional Verification of Real-Time UML Designs. In *Proc. of the European Software Engineering Conference (ESEC), Helsinki, Finland*, pages 38–47. ACM Press, September 2003.
4. T. Hestermeyer, P. Schlautmann, and C. Ettingshausen. Active suspension system for railway vehicles-system design and kinematics. In *Proc. of the 2nd IFAC - Confecence on mechatronic systems*, Berkeley, California, USA, 9-11December 2002.
5. Sven Burmester, Holger Giese, and Wilhelm Schäfer. Model-driven architecture for hard real-time systems: From platform independent models to code. In *Proc. of the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA'05), Nürnberg, Germany*, LNCS, pages 1–15. Springer Verlag, November 2005.
6. Henrik Ejersbo Jensen, Kim Guldstr, Kim Guldstr, and Arne Skou. Scaling up Uppaal Automatic Verification of Real-Time Systems using Compositionality and Abstraction. In *Proceedings of the 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 2000)*, volume 1926 of *LNCS*, Pune, India, September 2000. Springer Verlag.
7. Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. In *Formal Modeling and Analysis of Timed Systems: First International Workshop, FORMATS 2003, Marseille, France, September 6-7, 2003. Revised Papers*, volume 2791, pages 182 – 188, 2004.
8. Holger Giese and Martin Hirsch. Timed and Hybrid Refinement in Mechtronic UML. Technical Report tr-ri-03-266, University of Paderborn, Paderborn, Germany, December 2005.
9. Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, volume 3185 of *LNCS*, pages 200–236. Springer Verlag, September 2004.

[4] http://www.fujaba.de

10. Sven Burmester, Holger Giese, Martin Hirsch, and Daniela Schilling. Incremental design and formal verification with UML/RT in the FUJABA real-time tool suite. In *Proceedings of the International Workshop on Specification and vaildation of UML models for Real Time and embedded Systems, SVERTS2004, Satellite Event of the 7th International Conference on the Unified Modeling Language, UML2004*, October 2004.
11. Aditya Agrawal, Gyula Simon, and Gabor Karsai. Semantic Translation of Simulink/Stateflow models to Hybrid Automata using Graph Transformations. In *International Workshop on Graph Transformation and Visual Modeling Techniques, Barcelona, Spain*, 2004.
12. R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical Hybrid Modeling of Embedded Systems. In *First Workshop on Embedded Software*, 2001.
13. Thomas A. Henzinger. Masaccio: A Formal Model for Embedded Components. In *Proceedings of the First IFIP International Conference on Theoretical Computer Science (TCS)*, volume 1872 of *LNCS*, pages 549–563, 2000.
14. R. Grosu, T. Stauner, and M. Broy. A modular visual model for hybrid systems. In *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, volume 1486 of *LNCS*. Springer Verlag, 1998.
15. Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O Automata Revisited. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC 2001), Rome, Italy, March 28-30, 2001*, volume 2034 of *LNCS*, pages 403–417. Springer Verlag, 2001.
16. Leslie Lamport. Hybrid systems in tla+. In *Hybrid Systems*, pages 77–102, London, UK, 1993. Springer Verlag.
17. Thomas A. Henzinger, Marius Minea, and Vinayak Prabhu. Assume-Guarantee Reasoning for Hierarchical Hybrid Systems. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC 2001), Rome, Italy, March 28-30, 2001*, volume 2034 of *LNCS*, pages 275–290. Springer Verlag, 2001.
18. Thomas A. Henzinger, Christoph M. Kirsch, Marco A.A. Sanvido, and Wolfgang Pree. From Control Models to Real-Time Code Using Giotto. In *IEEE Control Systems Magazine 23(1):50-64, 2003. A preliminary report on this work appeared in C.M. Kirsch, M.A.A. Sanvido, T.A. Henzinger, and W. Pree, A Giotto-based helicopter control system, Proceedings of the Second International Workshop on Embedded Software (EMSOFT)*, volume 2491 of *LNCS*, pages 46–60. Springer Verlag, 2002.
19. Dirk Beyer. Efficient reachability analysis and refinement checking of timed automata using BDDs. In T. Margaria and T. F. Melham, editors, *Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME 2001)*, volume 2144 of *LNCS*, pages 86–91. Springer Verlag, 2001.