Modeling Collaborations with Dynamic Structural Adaptation in Mechatronic UML*

Martin Hirsch and Stefan Henkler
Software Engineering Group
University of Paderborn
Warburger Str. 100
D-33098 Paderborn, Germany
[mahirsch|shenkler]@uni-paderborn.de

Holger Giese
System Analysis and Modeling Group
Hasso Plattner Insitute
University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam, Germany
holger.giese@hpi.uni-potsdam.de

ABSTRACT

The next generation of advanced mechatronic systems is expected to behave more intelligently than today's systems. These systems are expected to enhance their functionality and improve their performance by building communities of autonomous agents which exploit local and global networking. Such mechatronic systems will therefore include complex coordination protocols which require execution in real-time and reconfiguration of the locally employed control algorithms at runtime to adjust their behavior to the changing system goals leading to self-adaptation. In this paper we will present an extension of the MECHATRONIC UML approach which will enable us to model collaborations between components which include structural adaptation and multi-ports. Besides the modeling of complex collaborations and the rules to join and leave these collaborations via ports and multi-ports, we propose to employ hierarchical state machines with a dynamic number of submachines to model the behavior of the multi-ports. For the collaborations this involves the related protocols, while for the components we have to refine this behavior to ensure a proper synchronization with other parts of the component behavior.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*model checking*; D.2.11 [**Software Engineering**]: Software Architectures—*domain-specific architectures*

General Terms

Design, Verification

*This work was partly 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.

SEAMS'08, May 12–13, 2008, Leipzig, Germany.

Keywords

Collaboration, Dynamic Structural Adaption, Mechatronic Systems, Safety

1. INTRODUCTION

It is expected that the next generation of advanced mechatronic systems will behave more intelligent than today's systems. These systems will improve their behavior with respect to functionality and performance by acting as communities of autonomous agents which use their networking to optimize and adjust their behavior. Complex coordination protocols which require execution in real-time and reconfiguration of the locally employed control algorithms at runtime to adjust their behavior to the changing system goals characterize these mechatronic systems.

We addressed in former works many of the the resulting challenges [9, 8, 11, 15, 6, 12, 7] with the model-driven MECHATRONIC UML development approach [9, 8] which combines domain specific modeling and refinement techniques with verification based on compositional model checking. The approach suggests modeling the software by using a refined UML 2.0 component model including the detailed definition of ports, connectors, and coordination patterns. The MECHATRONIC UML approach further refines the component model to define a proper integration between discrete and continuous control such that the reconfiguration of hierarchical component systems can be described in a modular way. However, the provided support for coordination patterns requires that the patterns have a fixed number of roles and cannot themselves change their structure.

As oftentimes the collaboration between a flexible number of participants is required, we extend in this paper our MECHATRONIC UML approach such that we can model collaborations between components which include structural adaptation in form of new or removed ports as well as multi-ports. The modeling of complex collaborations will be possible by means of rules which describe how to join and leave these collaborations via ports or multi-ports: hierarchical state machines with a dynamic number of submachines are further introduced to model the behavior of the multi-ports. For the collaborations they are employed to describe the multi-port protocols. For the components we use them to refine the role behavior to model a proper synchronization with other parts of the component behavior. To enable us to still use the compositional reasoning framework of the MECHATRONIC UML approach, we discuss also how these modeling techniques can be covered by related verification tasks that are able to cope with their more dynamic nature.

Application Example. A concrete example for a complex mechatronic product where the outlined need for collaborations with flexible number of participants is the RailCab R & D project[1]. The vision of the RailCab project is a mechatronic rail system where autonomous shuttles apply the linear drive technology, used in the Transrapid, but travel on the existing passive track system of the standard railway.

The particular problem considered in this paper (previously presented in [15]) is to coordinate the autonomously operating shuttles in such a way that they form convoys in order to reduce the energy consumption caused by air resistance and achieve a higher system throughput. Such convoys are established on-demand and require small distances between the shuttles. Since the decision of building and breaking a convoy during run-time, the main characteristics of such a system is self-adaptivity.

The required small traveling distances causes the real-time coordination between the speed control units of the shuttles to be a safety-critical aspect and results in a number of constraints, which have to be addressed when building the control software of the shuttles. Shuttles communicate with each other and their environment. Since the communication is wireless and unreliable, message loss and message delay is possible and in case of a network failure it is necessary to enforce a controlled emergency break. Also other specific emergency cases, e.g. if the linear drive module fails, have to be addressed.

In Figure 1 a sketch of the convoy coordination via ports is depicted. In our example, each shuttle is parametrized with a specific driving/break characteristic p_i.

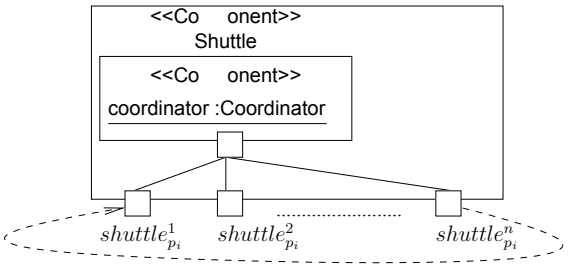

Figure 1: Sketch of convoy coordination via ports - each shuttle is parametrized with a specific driving/break characteristic p_i

Tour. The structure of the paper is as follows: We first review the original MECHATRONIC UML approach in Section 2. Then, we present the concepts for modeling collaborations with a flexible number of participants and multi-ports in Section 3. Afterwards the treatment of multi-ports with in the components are covered in Section 4. Finally, we will discuss relevant related work in Section 5 and present our conclusions and an outlook on future work.

2. FOUNDATIONS

The MECHATRONIC UML approach addresses self-optimizing mechatronic systems which exploit the today available advanced processing capabilities and local networking to optimize the behavior of its autonomous units, local groups, as well as the overall systems. Giese and Henkler discussed the general requirements for the model driven development of software-intensive (mechatronic) systems in [13]. Besides a modeling approach supporting appropriate abstraction, the integration of other disciplines, in our case

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

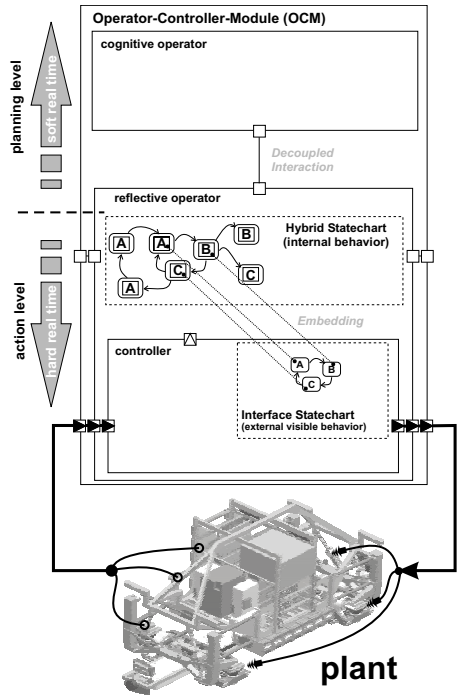

Figure 2: UML structure of the OCM architecture

the control engineering, is required. Further, an appropriate support for run-time adaption is required. In the next two sections, we will discuss the existing support of MECHATRONIC UML for software-intensive mechatronic systems. In the last section, we will conclude with the limitations of this approach.

2.1 Hierarchal Structuring

The overwhelming number of functions realized by a single autonomous unit makes appropriate structuring techniques imperative when designing the corresponding information-processing unit. Therefore, the MECHATRONIC UML approach employs the Operator-Controller-Module (OCM) [6] architecture, which is presented in Figure 2.

It decomposes the hierarchical architecture of a self-optimizing mechatronic unit as follows: (1) On the lowest level of the OCM, there is the *controller* (C) featuring an arbitrary number of alternative control strategies. Within the OCM's innermost loop, the currently active control strategy processes measurements and produces control signals. As it directly affects the plant, it is called *motor loop*. The software processing is necessarily quasi-continuous, including smooth switching between the alternative control strategies. (2) The controller is complemented by the *reflective operator* (RO), in which monitoring and controlling routines are executed. The reflective operator operates in a predominantly event-oriented manner. It does not access the actuators of the system directly, but may modify the controller and initiate the switch between control strategies. It furthermore serves as the connecting element to the cognitive level of the OCM. (3) The topmost level of the OCM is occupied by the *cognitive operator* (CO). On this level, the system can gather information concerning itself and its environment and employ it for the improvement of its own behavior.

To realize this architecture the controller can be modeled using CAE tools and block diagrams. The reflective operator and cognitive operator are addressed with MECHATRONIC UML offering a

real-time variants of state machines to describe the reflective operator and all UML concepts for non real-time behavior of the cognitive operator.

To support the modular reconfiguration of the internal structures of the controllers, MECHATRONIC UML provides the concept of hybrid UML components and a related hybrid Statechart extension for the UML [6]. The hybrid components support the design of self-optimizing mechatronic systems by allowing specification of the necessary flexible reconfiguration of the system as well as of its hybrid subsystems in a modular manner.

2.2 Coordination

To also address the coordination between the different autonomous mechatronic units MECHATRONIC UML support *coordination patterns* [15]. These patterns permit dividing the modeling into modeling the interaction between components of the system by means of the reusable coordination patterns and modeling the detailed behavior of the components by relating to the behavior of the applied patterns and their roles to corresponding component ports.

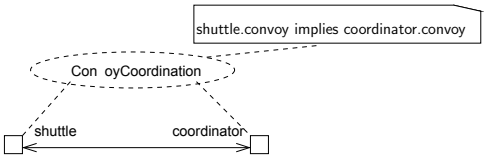

Figure 3: Simple coordination pattern

A pattern, as depicted in Figure 3, describes communication and therefore consists of multiple communication partners, called *roles*. Roles are linked by a connector. The communication behavior of a role is specified by a real-time statechart and is restricted by an invariant. The behavior of the connector is described by another real-time statechart that is used to model besides the transport of the messages the possible delay and the reliability of the channel, which are of crucial importance for real-time systems. The overall behavior of a pattern has to guarantee a defined pattern property, whereas the behavior of a role can be restricted by role invariants.

Within the example, coordination between two shuttles is modeled as a pattern. This ConvoyCoordiantion pattern consists of two roles, the shuttle role and the coordinator role and one connector that models the link between the two shuttles. The pattern specifies the behavior needed to coordinate two successive shuttles. The main requirement the pattern is addressing is ensuring that no collision can happen (shuttle.convoy implies coordinator.convoy).

Figure 4: Application of the simple coordination pattern

Figure 4 shows the application of the simple coordination pattern. In this example the rear shuttle *shuttle2* realizes the *shuttle* role and the front shuttle *shuttle1* behaves as a *coordinator*.

2.3 Limitations

While the presented approach provide the fundamental ingredients to model and also compositionally verify the correct and safe operation of self-optimizing mechatronic systems. However, the current capabilities include some severe restrictions. We can (1) only have a fixed number of roles in a coordination pattern while in several cases we have the requirement to organize the coordination between a variable number of autonomous units with given upper bound and (2) a component can only provide a fixed number of ports while support for models with multi-ports would be desirable for cases where on component is responsible for the coordination of a variable number of other components, as discussed in the introduction. In the following, we will discuss the extension of our MECHATRONIC UML approach for supporting this kind of requirements.

3. MODELING COLLABORATIONS

To overcome the limitations of the component and pattern based approach sketched in the last section, we introduce in this section *Dynamic collaboration* which support more complex communication structures described by multiple pattern instances and related reconfiguration rules. By this extension we can model and verify the envisioned convoy coordination.

Like the simple coordination patterns the complex patterns consists of roles and connectors. In Figure 5 the complex pattern for the ConvoyCoordination is depicted. One optional characteristic of such complex patterns are *multi-roles*. Multi-roles are depicted by overlapping ordinary ports associated with an cardinality and are a short hand for a set of ordinary roles belonging to the same component. In our ongoing example coordination is a complex port with cardinality n. The counterpart, the *shuttle* role is associated with cardinality 1. For this reason every "'subrole"' of the multi-role is connected to one unique *shuttle* role similar to a one to many association in a class diagram. To cope with the new complex behavior we have to extend the power of the pattern constraints/invariants such that collaboration constraints/invariants can define properties for a dynamically changing number of roles (cf. Figure 1).

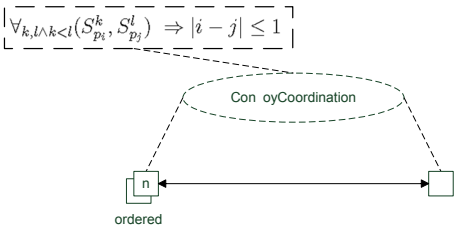

Figure 5: Collaboration with dynamic structural adaptation

As mentioned above the complex role is associated with the attribute {ordered}. From this follows, that all subroles are managed in a predefined order.

Figure 6 shows the *Shuttle* component. The component embeds the two components Coordinator and Velocity. The behavior of the component is defined as follows. The Shuttle component must conform to the ConvoyCoordination pattern and has to operate as both a coordinator and a shuttle as it may be followed by another shuttle as well as itself can follow another shuttle. Therefor, the component has a multi-port which is connected by an assembly with the inner component Coordinator and with the Velocity component.

In Figure 7 two shuttle instances which apply the ConvoyCoordination Collaboration with dynamic structural adaptation are shown.

Our idea is to describe all valid structural reconfiguration steps (e. g. if a shuttle join or leave a convoy) for a *Dynamic collaboration* by structural rules. First we give a brief overview about the underlying Story Pattern formalism. Thereafter, by means of the introduced convoy building example we explain all valid reconfiguration rules for the ConvoyCoordination Collaboration with dynamic structural adaptation.

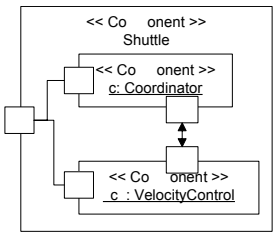

Figure 6: The type Shuttle

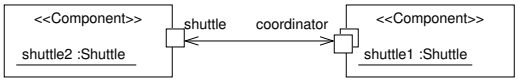

Figure 7: Instance view

Story Patterns are an extended type of UML object diagram (cf. [21]) that allow expressing properties and transformations, especially structural changes. A Story Pattern consists of two object diagrams representing a pre- and a postcondition, the left hand side (LHS) and the right hand side (RHS). At runtime, the LHS is matched against the instance graph, and the variables of the pattern are bound to specific nodes and edges. If a match is found, it is transformed in order to match the RHS by adding, modifying and deleting the appropriate nodes and edges using the Single Push Out strategy (SPO). When specifying Story Patterns, the RHS is integrated into the LHS in order to obtain a compact representation. This is achieved by using the stereotypes $++$ for marking exclusive elements of the RHS that need to be created and $--$ for denoting elements of the LHS which should be deleted as a side-effect of the rule.

In the following we explain the rules which describe all redonfiguration steps that are performed by the *Dynamic collaboration*. The rules are splitted into *inital rules*, *extension rules*, and reduction rules.

3.1 Structure

Initial Rule.

First of all we have to decribe the initial rule, which creates our shuttle instances. In Figure 8 this rules is depicted.

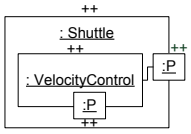

Figure 8: Initial rule

Extension Rules.

In case of the *Dynamic collaboration*, we have to distinguish between two *extension rules*. The first one (cf. Figure 9) describes the building of a convoy with length 2. Our assumption is that the front shuttle will always be the *coordinator* of the convoy. Therefore a instance of the Coordinator component is created as a subcomponent of the front shuttle. Also a connection between both shuttles is created. The stereotype ≪last≫ indicates that the depicted instance shuttle2 is the last shuttle connected to the convoy. The resulting instance situation which results from the application of the rule is depicted in Figure 10.

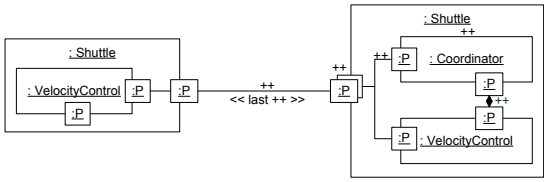

Figure 9: Extension rule 1

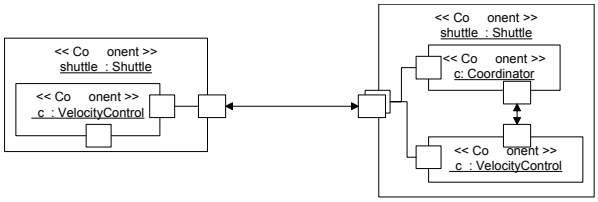

Figure 10: Instance view

In case of a shuttle aims to join an existing shuttle convoy we have to introduce a second extension rule (cf. Figure 11). This rules ensures that a new shuttle will join the convoy at the last position. By the ≪last≫ construct, this position can be identified unique. Besides creating a new connection between the last two shuttle the ≪last≫ construct has also to be deleted and linked to the new connection. The instance situation after applying the extension rule is depicted in Figure 12.

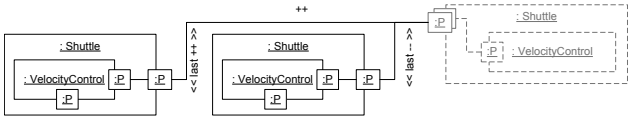

Figure 11: Extension rule 2

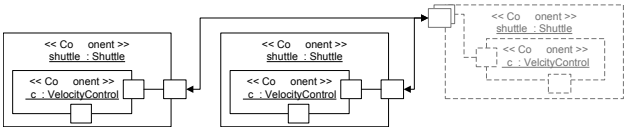

Figure 12: Instance view

Reduction Rules.

In the last paragraph we have introduced the rules which allows us to specify the buildup of a convoy. Now, for completeness we introduce the *reduction rules* which describe the breakup of a convoy or at least the leaving of one shuttle of a convoy. In order to do not change the convoy order when a reduction rule is applied, similar to the creation rules we use the matching of the ≪last≫ construct to figure out which shuttle can be removed from the convoy (cf. Figure 13). If the convoy only exists of two shuttle anymore we have to breakup the convoy totaly. This causes the deactivation of the coordinator in the leader shuttle. Therefore besides deleting

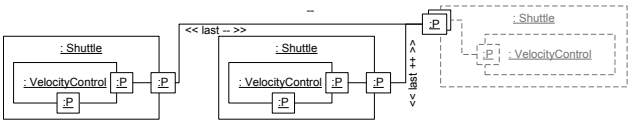

Figure 13: Last shuttle leaves a convoy

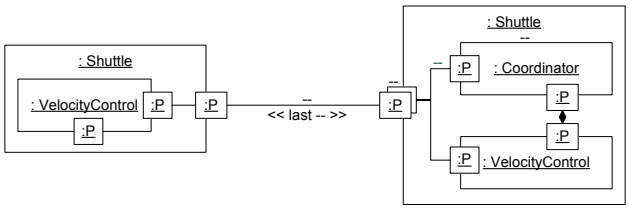

Figure 14: Reduction rule 2: Convoy of length 2 is splitted

the connection between both shuttle the subcomponent Coordinator is deleted as well (cf. Figure 14). Last, we have to explain the rule if the leader shuttle leaves the convoy. For that reason we have to "'pass"' the Coordinator component to the second shuttle, delete the connection and create a subcomponent Coordinator in the second (now leader) shuttle (cf. Figure 15).

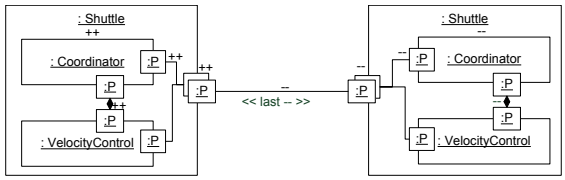

Figure 15: Reduction rule 3: Front shuttle leaves a convoy

3.2 Behavior of Roles

In the last section we have introduced all graph-based rules which describe the buildup and breakup of a shuttle convoy. These rules only refer to the structural changes. Now, we address the behavior description of the involved roles of the *Dynamic Adaptation*.

The new parametrized role *coordinator* of the *Dynamic Adaptation* is depicted in Figure 16. The extions w. r. t. to the former role statechart are the synchronization channels next_k and $\mathsf{nextFailed}_k$. By this channels each separate role synchronizes with the adaption role statechart (cf. Figure 17). In more detail the real-time state-

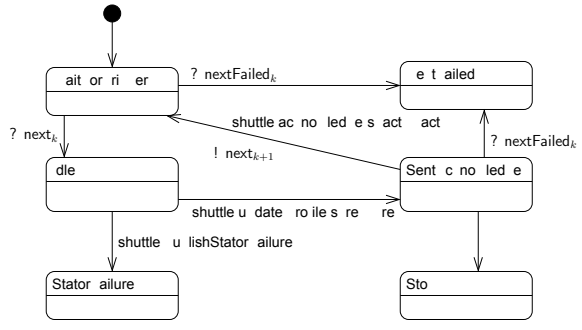

Figure 16: The behavior of a coordination role role$_k$

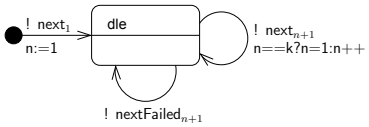

Figure 17: The adaption role statechart for all *coordinator* roles

chart of the *shuttle* role consists of three states. Initially the role is in state *Normal*. Every 150 time units the role has to received an *update* message from the *coordinator* role. This message includes the current *profile*, reference position and reference velocity. The role confirms the message with an *acknowledge* message including the actual position and actual velocity. In case of no *update* message is received (e.g. network failure) the role switches after 150 time units to the *network failure* state. The state *StatorFailure* will be reached if another shuttles propagates an StatorFailure.

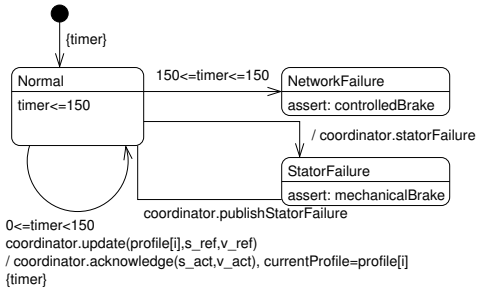

Figure 18: The behavior of a shuttle role shuttle

3.3 Verification

If we want to address the verification problem for the outlined system, we have to face two challenges. At first, we have to integrate the continuous control behavior with the event-based hard real-time coordination which is an inherent characteristics of the considered domain and problem class. In addition, in the considered class of systems the coordination takes places between a variable number of participants, i.e., a convoy consist of a variable number of shuttles which can be modeled with complex patterns. We can use similar techniques as discussed in [15] as long as the upper bound of the number of involved roles is small enough. We therefore have to first expand the potentially infinite state model of the collaboration into one where the structural adaptation steps are encoded in finite state timed automata using the given upper bounds on the number of roles. We can then verify that the collaboration constraints/invariants hold for the collaboration behavior.

4. COMPONENT SPECIFICATION

Components are designed by the collaboration patterns as presented in the previous section. Components apply this abstract collaboration patterns by refining the behavior of the applied role. The refinement has to respect the role automaton by not adding possible behavior or blocking guaranteed behavior. Additionally, the refinement has to respect the guaranteed behavior of the roles in form of its invariants (cf. [15]). An additional internal statechart for synchronization is used to describe the required coordination (cf. Figure 19) between the applied roles. As we introduce an additional coordination layer for the role behavior to enable multi ports, the synchronization is realized with the internal statechart and this coordination layer or, if there is just only one port, with the role behavior directly. Hence, the coordination layer capsules the access

to the ports to focus on the relevant parts required for the synchronization. In the following, we do not consider explicitly the structural effects of creation and reduction rules. The velocity controller component is omitted (implicitly realized by every shuttle) and the coordinator component is realized by a coordinator sub-statechart (cf. Figure 20).

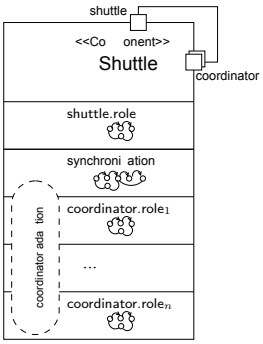

Figure 19: Shuttle with adaptive behavior

In our example, the shuttle component must conform to the convoyCoordination pattern and has to operate as a coordinator or as a shuttle. The non-deterministic choice of proposing, accepting, and coordinating a convoy is specified by the additional synchronization statechart of the role refinements (Figure 20). The port statecharts which refine the pattern roles are shown in Figure 21 and Figure 22.

4.1 Synchronization

The shuttle synchronization statechart initiates the building of a coordinated convoy by sending start to the refined shuttle role and a createPort to the refined coordination role when the guards indicate that it is useful to run in convoy mode and the decision in which role the shuttle should be is evaluated. The initial event of building a convoy or the decision in which role a shuttle should be is made by a cognitve operator, e.g by a track section control which knows all shuttles and their order (cf. events convoyUseful, coordinator, and shuttle). The coordinated convoy is broken or restored, when a failure is manifested. The roles identify this situation, when a message is not received in time. Then, the synchronization statechart triggers the acceleration control with the specific profile of the shuttle. The recovery of the coordinated convoy is not considered in more detail.

4.2 Refine Multi-Roles

The refinement of the coordination role is done only by the adaption statechart. The coordination role is initialized by the createPort send by the synchronization statechart. CreatePort(shuttle) instantiates a new coordination role. When the adaption statechart is in idle additionally a port could be added, initiated by the synchronization statechart, too. As mentioned before, if the update routine of a coordination role fails, the synchronization statechart is informed to restore the convoy. The recovery process is considered any more. In principle, the shuttles have to identify if a convoy is still possible with less shuttles, the synchronization statechart has to delete the defect shuttles/ports to the shuttles, and the ordering have to be restored.

The refinement of the shuttle role is similar to that of the coordination role. The initialization of this role is triggered by the synchronization statechart (by sending start). If a failure is man-

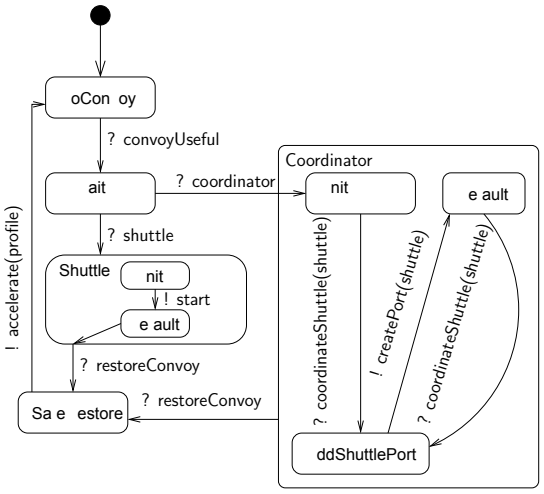

Figure 20: Shuttle synchronization statechart

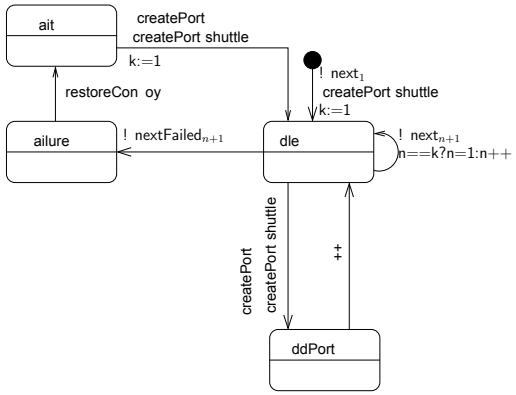

Figure 21: Refined adaption statechart for coordination roles

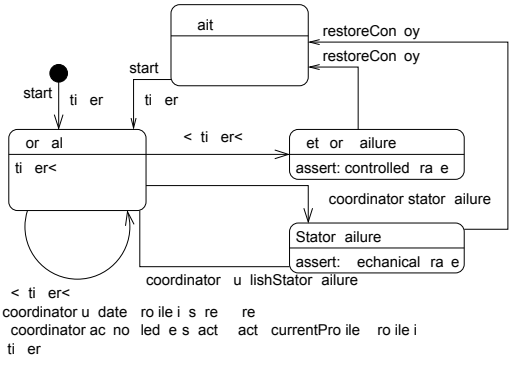

Figure 22: Refined shuttle role

ifested the synchronization is informed to restore the convoy. In case of shuttle role the decision to take is whether the shuttle could be in a (smaller) convoy or no convoy is possible.

In the shown example, the synchronization statechart considers all coordination roles in the same manner. Hence, the adaption statechart capsules all coordination roles. If we consider other examples, like an online real-time auction, the coordination role to each shuttle would be the same, but each shuttle is considered different. Hence, the interface to the coordination is getting more complex and further, creating a port to a new shuttle will add additional behavior to the synchronization statechart. This could be realized by similar techniques as discussed in the previous section by adding additional components representing the required specific behavior for each created port.

4.3 Verification

Also in case of the component specification and multi-ports we can use the techniques presented in [15] for verification exploiting that the upper bound of the number of involved multi-ports is small enough. Like in the case of the collaborations we have to first expand the model into one where the structural adaptation steps are encoded in finite state timed automata and can then verify that the role invariants hold for the component behavior and that it also respects each role automaton. To ensure that the port refines each of the role protocols associated to its ports, we propose to use syntactical refinement rules instead of an explicit verification step.

5. RELATED WORK

The *de facto* industry standard for modeling of mechatronic systems with hybrid behavior is MATLAB/Simulink and Stateflow[2]. Formal verification of MATLAB/Simulink and Stateflow models of moderate size can be accomplished by automatically transforming them to hybrid automata (cf. [1]). However the verification does not scale to real world examples. In the following we first review some approaches covering the problem of verifying UML models. In the remainder we discuss approaches for verification of hybrid systems.

Beyond UML and its profiles, a number of proprietary approaches for the modeling and verification of technical systems with UML exist.

Knapp et al. present in [20] a tool called HUGO/RT. Within this tool, models are described by UML state machines. The properties to be checked are given as scenarios written as sequence diagrams extended by time annotations. There is no support for hybrid pattern and pattern decomposition.

The aim of the IST OMEGA project [16] is to ensure the correctness of embedded systems. In the approach, the UML has been extended by additional time constructs and a formally defined semantics is intended. However, unlike our approach, there is no support for hybrid behavior and compositional verification. Verification is only supported for the semi-automatic verification via theorem proving.

Existing model checkers for hybrid behavior (cf. survey [14]) are rather limited with respect to the supported hybrid system classes. While the model checking problem is only decidable for the rather restricted class of rectangular automata which at most support constant first derivations, for more expressive continuous models up to now only approximation algorithms exists. Even these most expressive models require that all parameters of the model are set at design time while very often situation specific parameters for the requested continuous behavior (e.g., the trajectory of the shuttles) are used in practice when highly adaptive systems are considered (parameter adaptation).

A hierarchic automata model for the specification of behavior is provided by the modeling language CHARON [3][19]. Additionally, it provides a hierarchical architectural model, based on ROOM actor diagrams. [4] defines *refinement* for hybrid CHARON models. CHARON's focus is the formal verification of hybrid systems based using discrete abstraction based on predicate abstraction.

Within the Fresco project, the high-level modeling language Masaccio [17] has been developed. It builds complex components by the parallel and serial composition of atomic discrete and atomic continuous components. The basis of Masaccio is the formal semantics for the domain of hybrid dynamical systems. The focus of Masaccio is to provide a formalized modeling language for verification.

Automatic verification of the real-time behavior including the reconfiguration is supported by CHARON, Masaccio and MECHATRONIC UML. CHARON and Masaccio even supports model checking of hybrid models. MECHATRONIC UML and Masaccio support compositional and modular model checking of real-time properties employing refinement[3] which is also possible for CHARON and HyCharts in principle as they also support a notion of refinement. CHARON additionally supports predicate abstraction as a means to improve the scalability of the verification. per!

There are some approaches for modeling the structural aspects of adaptive systems [26, 24, 18, 25, 23] or the behavioral aspects [27, 2, 22, 10] but non of them consider both aspects [5].

6. CONCLUSION AND FUTURE WORK

In this paper we presented an extension of the model-driven MECHATRONIC UML approach for self-optimizing mechatronic systems. A more general notion of collaborations with a dynamically changing number of participants (roles), the extension of components towards a dynamically changing number of ports by means of multi-ports, and first idea for the proper verification of safety properties for the resulting models have been presented. This includes an extension of the employed modeling techniques for the different modeling artifacts as well as an extension of the employed architectural patterns. We have demonstrated that the proposed extensions enable the modeling of more complex self-organizing behavior in form of collaborations which extent the capabilities of the former invented coordination patterns. Also the hierarchical component concepts for the safe self-optimization at several levels of the hierarchy has been extended and the top-level component can now support changes in the number of ports in the case of multi-ports.

In future work it is planned to consider the sketched verification problem in a more appropriate manner such that also collaborations with a higher number of roles can be tackled. Further, we plan to generalize the approach and try to apply the concepts to other domains, e. g. the service oriented architecture domain.

7. REFERENCES

[1] A. Agrawal, G. Simon, and G. 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.

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

[3] The definition of *refinement* is informally as follows: A model *refines* another one, if it behaves always compliant with the behavior of the original model.

[2] R. Allen, R. Douence, and D. Garlan. Specifying and analyzing dynamic software architectures. *Lecture Notes in Computer Science*, 1382:21–36, 1998.

[3] 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.

[4] R. Alur, R. Grosu, I. Lee, and O. Sokolsky. Compositional Refinement of Hierarchical Hybrid Systems. In *Proceedings of the Fourth International Conference on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 33–48. Springer Verlag, 2001.

[5] J. S. Bradbury, J. R. Cordy, J. Dingel, and M. Wermelinger. A survey of self-management in dynamic software architecture specifications. In *WOSS '04: Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pages 28–33, New York, NY, USA, 2004. ACM.

[6] S. Burmester, H. Giese, and O. Oberschelp. Hybrid UML Components for the Design of Complex Self-optimizing Mechatronic Systems. In H. Araujo, A. Vieira, J. Braz, B. Encarnacao, and M. Carvalho, editors, *Proc. of 1st International Conference on Informatics in Control, Automation and Robotics (ICINCO 2004), Setubal, Portugal*, pages 222–229. INSTICC Press, August 2004.

[7] S. Burmester, H. Giese, and O. Oberschelp. Hybrid UML Components for the Design of Complex Self-optimizing Mechatronic Systems. In *Informatics in Control, Automation and Robotics*. Kluwer Academic Publishers, 2005. to appear.

[8] S. Burmester, H. Giese, and M. Tichy. Model-Driven Development of Reconfigurable Mechatronic Systems with Mechatronic UML. In U. Assmann, A. Rensink, and M. Aksit, editors, *Model Driven Architecture: Foundations and Applications*, LNCS, pages 1–15. Springer Verlag, 2005.

[9] S. Burmester, M. Tichy, and H. Giese. Modeling Reconfigurable Mechatronic Systems with Mechatronic UML. In U. Aßmann, editor, *Proc. of Model Driven Architecture: Foundations and Applications (MDAFA 2004), Linköping, Sweden*, pages 155–169, June 2004.

[10] C. Canal, E. Pimentel, and J. M. Troya. Specification and refinement of dynamic software architectures. In *WICSA1: Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, pages 107–126, Deventer, The Netherlands, The Netherlands, 1999. Kluwer, B.V.

[11] H. Giese. A Formal Calculus for the Compositional Pattern-Based Design of Correct Real-Time Systems. Technical Report tr-ri-03-240, Lehrstuhl für Softwaretechnik, Universität Paderborn, Paderborn, Deutschland, July 2003.

[12] H. Giese, S. Burmester, W. Schäfer, and O. 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.

[13] H. Giese and S. Henkler. A survey of approaches for the visual model-driven development of next generation software-intensive systems. In *Journal of Visual Languages and Computing*, volume 17, pages 528–550, December 2006.

[14] H. Giese and S. Henkler. A survey of approaches for the visual model-driven development of next generation software-intensive systems. *J. Vis. Lang. Comput.*, 17(6):528–550, 2006.

[15] H. Giese, M. Tichy, S. Burmester, W. Schäfer, and S. 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.

[16] S. Graf and J. Hooman. Correct Development of Embedded Systems. In F. Oquendo, B. Warboys, and R. Morrision, editors, *Proceedings of the First European Workshop on Software Architecture, EWSA2004*, volume 3047 of *Lecture Notes in Computer Science*, pages 241–249, St Andrews, UK, May 21-22 2004. Springer Verlag.

[17] T. A. Henzinger. Masaccio: A formal model for embedded components. In *Proceedings of the First IFIP International Conference on Theoretical Computer Science (TCS), Lecture Notes in Computer Science 1872, Springer-Verlag, 2000, pp. 549-563.*, 2000.

[18] D. Hirsch, P. Inverardi, and U. Montanari. Graph grammars and constraint solving for software architecture styles. In *ISAW '98: Proceedings of the third international workshop on Software architecture*, pages 69–72, New York, NY, USA, 1998. ACM.

[19] F. Ivancic. *Modeling and Analysis of Hybrid Systems*. PhD thesis, University of Pennsylvania, 2003.

[20] A. Knapp, S. Merz, and C. Rauh. *Model Checking timed UML State Machines and Collaborations*. 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT 2002), Oldenburg, September 2002, Lecture Notes in Computer Science volume 2469 pages 395-414. Springer-Verlag, 2002.

[21] H. Köhler, U. Nickel, J. Niere, and A. Zündorf. Integrating UML Diagrams for Production Control Systems. In *Proc. of the 22^{nd} International Conference on Software Engineering (ICSE), Limerick, Irland*, pages 241–251. ACM Press, 2000.

[22] J. Kramer and J. Magee. Analysing dynamic change in software architectures: A case study. In *CDS '98: Proceedings of the International Conference on Configurable Distributed Systems*, page 91, Washington, DC, USA, 1998. IEEE Computer Society.

[23] J. Kramer, J. Magee, and M. Sloman. Configuring distributed systems. In *EW 5: Proceedings of the 5th workshop on ACM SIGOPS European workshop*, pages 1–5, New York, NY, USA, 1992. ACM.

[24] D. L. Métayer. Software architecture styles as graph grammars. In *SIGSOFT '96: Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering*, pages 15–23, New York, NY, USA, 1996. ACM.

[25] P. Oreizy, N. Medvidovic, and R. N. Taylor. Architecture-based runtime software evolution. In *ICSE '98: Proceedings of the 20th international conference on Software engineering*, pages 177–186, Washington, DC, USA, 1998. IEEE Computer Society.

[26] G. Taentzer, M. Goedicke, and T. Meyer. Dynamic change management by distributed graph transformation: Towards configurable distributed systems. In *TAGT'98: Selected papers from the 6th International Workshop on Theory and Application of Graph Transformations*, pages 179–193, London, UK, 2000. Springer-Verlag.

[27] J. Zhang and B. H. C. Cheng. Model-based development of dynamically adaptive software. In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pages 371–380, New York, NY, USA, 2006. ACM.