Visual Integration of UML 2.0 and Block Diagrams for Flexible Reconfiguration in MECHATRONIC UML *

Sven Burmester† and Holger Giese
Software Engineering Group, University of Paderborn
Warburger Str. 100, D-33098 Paderborn, Germany
{burmi|hg}@uni-paderborn.de

Abstract

Today, complex, networked, self-adaptive mechatronic systems which integrate advanced control engineering and software engineering concepts within a single software system are envisioned. These systems adapt their structures at runtime to react to detected environmental changes, to change their system goals, or to react to a change of the system structure. To enable the development of such systems, an integration of object-oriented modeling techniques such as UML and control theory approaches such as functional block modeling is required. Thereby, the successful visual modeling concepts of control engineering should be preserved, as otherwise wide acceptance in industry, which is mainly dominated by control engineers, is very unlikely. In this paper, we present such a visual integration for UML 2.0 components, Statecharts, and block diagrams developed within the MECHATRONIC UML *approach. It permits to graphically model reconfiguration between several pre-defined configurations with statecharts and instance diagrams as well as to specify the flexible assembly of control configuration if needed by means of visual reconfiguration rules.*

1. Introduction

Today, mechatronic products have to integrate advanced control engineering and software engineering concepts within a single software system to enable that its elements operate flexible and self-adaptive within a complex networked environment. Such an integration has to enable the elements to react to detected environmental changes, to change their system goals, or to react to a change of the system structure by adapting their structure at runtime.

An integration between object-oriented modeling techniques such as UML and control theory approaches such as functional block modeling is required to enable the development of such systems (cf. the OMG effort for the integration of the software engineering domain with the control engineering domain [20]). Visual modeling concepts which preserve where possible the successful control engineering and software engineering concepts and integrate them in an intuitive graphical manner are required as otherwise wide acceptance in industry is very hard to achieve.

In this paper, we present our visual integration of the control engineering and software engineering worlds. UML 2.0 components and Statecharts as software engineering artifacts are integrated with block diagrams the classical control engineering notation within the model-driven MECHATRONIC UML development approach [6, 9, 8].

A first proposed integration suggests to model simple reconfiguration steps which allow to switch between several pre-defined configurations by extending UML components and Statecharts. A modular description with an intuitive visual appearance enables an easily comprehensible specification style for run-time reconfiguration. If a more flexible assembly of the control structure is required, we suggest to use visual reconfiguration rules to describe required reconfiguration steps which modify the current control configuration accordingly.

The paper outline is as follows: We first introduce a case study in Section 2 and afterwards, we review the current state of the art and identify several severe limitations when it comes to self-adaptive behavior and complex real-time coordination in Section 3. Then, we outline and discuss our proposal for the modular visual description of complex reconfiguration behavior using the case study in Section 4. The handling of flexible reconfiguration behavior to react to changes of the system structure follows in Section 5 before we close that paper by presenting our final conclusion and an outlook on planned future work.

*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 International Graduate School of Dynamic Intelligent Systems. University of Paderborn

2. Example

Our application example is taken from the RailCab project.[1] In this project, a modular rail system is developed consisting of autonomous shuttles which apply the linear drive technology used in the Transrapid,[2] but use existing rail tracks.

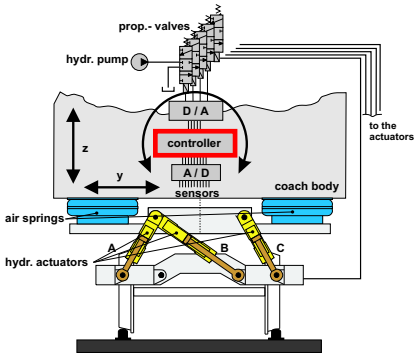

Figure 1. Suspension/tilt module

The shuttle's active suspension system and its optimization is one example for a complex mechatronic system we employ in the following. The suspension/tilt module, depicted in Figure 1, is based on air springs which are damped actively by a 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 and stability when controlling the shuttle's coach body. In order to achieve this goal, multiple feedback controllers are applicable with different capabilities in matters of safety and comfort.

In our controlling component, we apply the three feedback controllers reference, absolute, and robust, providing different levels of comfort and requiring different inputs. The most sophisticated controller reference uses a given trajectory $z_{ref} = f(x)$ that describes the ideal motion of the coach body and the absolute acceleration $\ddot{z}_{abs}$ of the coach body. The z_{ref} trajectory is given for each single track section and is communicated by a track section's registry to the shuttle. In case the reference trajectory is not available, the less comfortable controller absolute which requires only the $\ddot{z}_{abs}$ signal has to be used. In case the sensor that provides the $\ddot{z}_{abs}$ signal fails, the robust controller which provides the fewest comfort, but guarantees stability even when only standard inputs are available, has to be applied.

Another considered example is the control of shuttle convoys. Whenever suitable, the shuttles reduce the air resistance and thus reduce their energy consumption by building convoys as depicted in Figure 2. Such convoys are built on-demand and require a small distance between the different shuttles such that a high reduction of energy consumption is achieved. Coordination between speed control units of the shuttles becomes a safety-critical aspect and results in a number of hard real-time constraints, which have to be addressed when building the control software of the shuttles. As shuttles may show rather different characteristics due to their intent (shuttles for goods or people) or fabrication company, setting up the control structure for the convoy coordination requires to flexibly add shuttle specific control elements which might be provided at runtime (cf. compositional adaptation [17]).

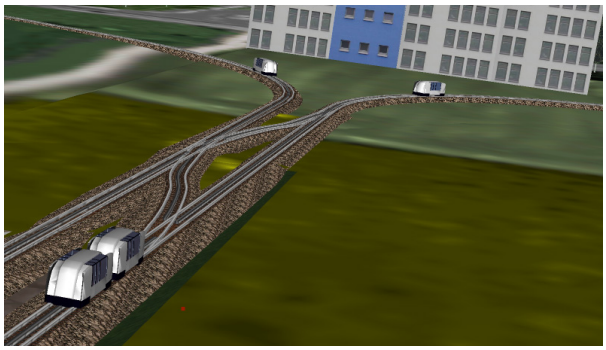

Figure 2. Shuttles building convoys

The challenges of modern mechatronic systems can be exemplified by referring to the introduced case study: (1) At first, sophisticated control algorithms have to be modeled to describe the high comfort control of the suspension/tilt module and thus support of CAE tools and their standard block diagram notations and libraries is required. (2) Secondly, the hard real-time coordination between the shuttles and the track section's registry best modeled with a suitable UML variant which supports real-time systems such as MECHATRONIC UML is a major concern and requires appropriate CASE tool support. (3) The required integration must further support to model the runtime reconfiguration of the controllers such that reconfiguration steps may either be initiated due to local events or due to changes of the current state of the hard real-time coordination with the external world, e.g. when the required reference trajectory has been received. (4) Furthermore, some reconfiguration steps can be realized as *atomic switching*, while sometimes a technique called *output fading* has to be employed. This technique fades the output of one controller out, while the output of another one is faded in to avoid discrete jumps which can lead to instabilities when switching between different feedback-controllers. (5) Finally, besides the preplanned reconfiguration, which switches between different control algorithms, also more flexible reconfiguration steps have to be supported to handle compositional adaptation as required, for example, in the case of the shuttle convoys.

[1] http://www-nbp.upb.de/en
[2] http://www.transrapid.de/en

3. State of the Art

To discuss the current state of the art and the limitations of current approaches, we first discuss block diagrams and hybrid automata before turning our attention to concepts which support a decomposition of the models such as hierarchical blocks or components.

Block diagrams [22] are the state of the art approach to specify feedback-controllers which is employed in all CAE tools such as Matlab/Simulink.[3] Discrete elements can be used in the block diagram to model reconfiguration of the feedback-controller structure (often using a Statechart like notation such as Stateflow in Matlab/Simulink). A first approach is to embed the discrete control elements into block diagrams in such a manner, that the discrete block, described by a statechart like notation, determines which one of a set of alternative controller outputs is let through. Thus, atomic switching between the output signals of different controllers can be directly modeled. To address output fading, an additional generator for the fading function g and a weighted output fading element $y = g(t)u_1 + (1 - g(t))u_2$ has to be controlled by the statechart. Another option to model reconfiguration are conditionally blocks which are only evaluated if explicitly triggered. Thus, a statechart can be used to only trigger the required elements of the currently active configuration instead of blinding out the results of the not required ones. The more formal hybrid bond graphs approach [18, 19] permits to blind out single components by so called controlled junctions, similar to discrete blocks in block diagrams.

From a visual language perspective, both approaches to describe reconfiguration become problematic if all five identified challenges have to be addressed. All configurations of required block diagrams have to be specified within a single complex diagram. Therefore, identification and comprehension of the different configurations becomes very difficult. The problems become even worse when it comes to the discrete control elements which describe the switching between the different configurations. Either they are rather unsystematically distributed in the block diagram as in the case of hybrid bond graphs or we end up with very complex statecharts describing the reconfiguration for the whole block diagram. The crucial problem here is that a consistent design of the different block diagram configurations and the statechart requires that the designer is able to identify the relation between control states of the statechart and block diagram configurations. However, the provided notations do not support such an understanding on the visual level.

The discussion of block diagrams and hybrid bond graphs showed that there is a strong relation between the system's current global discrete state and the current configuration. In the mentioned approaches, there is no direct support for a mapping which assigns to each discrete state a related configuration.

Hybrid automata [11, 1] overcome this drawback by simply assigning a specific continuous controller to each discrete state, so that each possible configuration is easily derived from the model without complex analysis. Extensions such as Hybrid I/O automata [16] further support communicating hybrid automata. Support for concepts such as hierarchic and orthogonal discrete states as known for statecharts have been introduced for Hybrid statecharts as defined in [12].

Although these approaches separate the possible configurations and thus overcome one of the drawbacks of block diagrams, the current proposals restrict the possible configurations assigned to a state to read the same inputs and produce the same outputs. Therefore, the interface –especially the information which input signals are required for a safe application of the controlller– is not present in these models. Another limitation, these approaches [11, 1, 16, 12] have in common is that they are restricted to the specification of behavior while an integration with an architectural description (similar to UML component diagrams or UML classes diagrams) is not provided. Therefore, these models can only be used for the specification of single components (in terms of UML), but a distributed system or a system with a modular, hierarchic architecture cannot be described.

If so called *hierarchic blocks* are employed in block diagrams to decompose the model, the visual complexity and problems to identify and comprehend the different configurations as well as their relation to discrete control states are less critical. However, this is only true when the interface of each hierarchic block is static and thus reconfiguration is restricted to happen only locally within the blocks. If this is not the case (not all inputs of a block are always required and not all output signals are always produced), the situation becomes even harder as the hierarchic blocks effectively hide their details and thus a designer cannot keep track of the configuration effects which can cross the block interfaces. It is to be noted that in order to address challenge (3), we thus either end up in the outlined dilemma that information which is required to comprehend the effects of reconfiguration are hidden or all effects of reconfiguration due to local as well as external effects has to be modeled without a hierarchical decomposition. While in the former case the decomposition becomes a hindrance for comprehension, in the latter case the complexity of the flat model would render any attempt to develop a thorough understanding of the reconfiguration.

Other approaches combining components and hybrid automata concepts such as CHARON [2], HyROOM [25, 3], HyChart [10, 24], HybridUML [4], and Ptomely II [15] provide hierarchical automata models for the specification of

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

behavior and hierarchical architectural models. In UML^h [7], the architecture is specified by extended UML classes diagrams that distinguish between discrete, continuous, and hybrid classes. Also the OMG effort to integrate models from the software engineering domain with models from the control engineering domain [20] falls into this category. The Systems Modeling Language (SysML) [21] is a first proposal to standardize system engineering, which could be integrated with a possible UML 2.0 successor (cf. [13]).

Like the hybrid automata and statecharts, all class- or component-based approaches also assume static interfaces. The main improvement in modeling is the introduction of a hierarchical architectural model. The behavior of the components of the architectural models can then be specified by the (hierarchical) behavioral models. Hybrid behavior is specified by adding continuous components in form of a MATLAB model, a differential equation, or a similar textual description to each discrete state of the component. Thus, the same limitations which have been identified for hierarchic blocks with static interfaces also apply here.

The preceding discussion highlights that current approaches are not sufficient to address the identified five challenges from Section 3: HyROOM and Ptomely II are the only approaches following challenge (1) and allowing the engineers to specify the continuous behavior with the well-known block diagrams. None of the discussed approaches allows an appropriate specification of real-time behavior (conf. challenge (2)) as their semantics –if defined– assume to detect triggered transitions and to fire them without consuming time which is unrealistic and not realizable. Further, they lack of reconfiguration based on local and external events (challenge (3)), as such a reconfiguration usually leads to a change of the interfaces and requires reconfiguration via multiple hierarchical levels, because the feedback-controller components are usually located on the lowest hierarchical level while the real-time coordination with the external world is usually at the higher ones. Current approaches allow just reconfiguration via one hierarchic level without changing the interface. Support for advanced visual constructs for a simple and intuitive specification of atomic transitions and fading transitions (challenge (4)) is not addressed at all. In addition, none of the approaches permits to handle flexible reconfiguration including compositional adaptation (challenge (5)). Instead, the anticipated reconfiguration steps have to be explicitly modeled right like the reconfiguration between two control algorithms.

4. Structural Reconfiguration

In this section, we present our approach to specify the control software for the suspension/tilt module which overcomes the drawbacks of the approaches w.r.t. challenges (1) to (4), discussed in the last section. We restrict our attention here to the visual language aspect. The underlying semantical integration with block diagrams has been discussed in [5] and the modular verification of the model consistency has been presented in [8] in detail.

When modeling the software to control the suspension/tilt module, we typically begin with a description of the system's structure, as depicted in the refined UML component diagram from Figure 3. The shuttle's Monitor component communicates with the Registry component of the upcoming track section to obtain the trajectory. The communication protocol is specified by the Monitor-Registration pattern, as described in [8]. If Monitor obtains the trajectory, it stores it in the storage component. The sensor is designed redundant: Three instances of the Sensor components provide the $\ddot{z}_{abs}$ signal. The CrossChecker judges all signals and determines if a sensor failed. The triangles in Figure 3 designate so called *continuous ports*. White ports are not always present – their existence is dependent on the components' discrete state. Note that the connections of the two black ports of BC are omitted in the example. Their connections do not change due to reconfiguration.

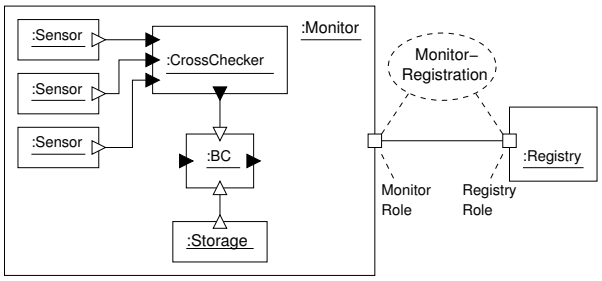

Figure 3. System structure

One task of the Monitor component is to coordinate its subordinated (embedded) Body Control (BC) component which contains the different feedback-controllers, controlling the suspension/tilt module. The behavior of Body Control is specified by our so called *hybrid reconfiguration chart* [8, 5] (see Figure 4). This hybrid reconfiguration chart consists of the three discrete locations Robust, Absolute, and Reference, each associated with one feedback-controller. Transitions that are visualized as thick arrows are associated with a deadline interval d_i, specifying a minimal and a maximal duration of the transitions, and the *fading function* f_{fade_i} describing how to fade. Such a fading transition is a visual construct to reduce complexity by omitting an intermediate location associated with a configuration that performs the fading. The other transitions fire *atomic*.

This example shows that the Body Control component has a dynamic interface, which depends on the current discrete state. Therefore, we provide a visual description of this dynamic interface in form of our so called *interface state chart* (see Figure 5) [8, 5]. This interface state chart

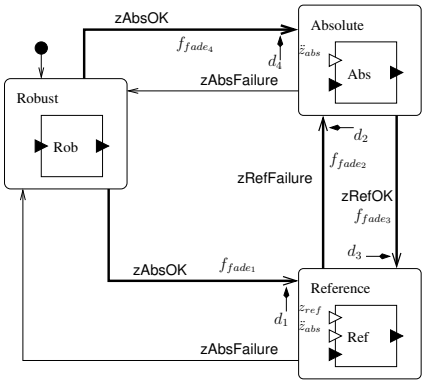

Figure 4. Behavior of the BC component

abstracts from the internal details of the behavior and depicts just the possible interfaces and the real-time restrictions (i.e. the deadline intervals) which describe how to switch between the interfaces. Especially, when the according hybrid reconfiguration chart contains multiple discrete states with different controllers, which have partly the same interface, the interface state chart provides an abstraction omitting much of the complexity of the reconfiguration chart. This abstract view of Body Control is used for the coordination with Monitor (see below).

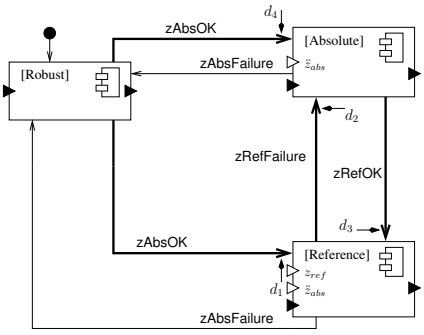

Figure 5. Dynamic Interface of the Body Control (BC) component

The Body Control component has to be coordinated by its superordinated Monitor component, dependent on the available input signals (z_{ref} and $\ddot{z}_{abs}$). Therefore, the description of the behavior of Monitor includes visually the subordinated components (see Figure 6).

The upper orthogonal state of the hybrid reconfiguration chart from Figure 6 contains four discrete states, representing that both, none, or exactly one of the input signals are available. The component instance diagrams, associated with each discrete state, specify the structure of the embedded components: For example, when Monitor is in state AbsAvailable, representing that just the $\ddot{z}_{abs}$ signal is available, it is specified that the Body Control component is in state Absolute and its input is fed by the CrossChecker component. A state change of Monitor to AllAvailable *implies* (i) a state change of Body Control to state Reference *and* (ii) a change of the structure of Monitor's embedded components, as indicated by the component instance diagram of the target state. It is to be noted that this relation can also be employed to check consistent reconfiguration at the interface level as outlined in [8] such that the Body Control's interface changes have to respect its interface state chart.

One of the significant properties of this approach is that control engineers *and* software engineers can continue using their well-known description languages and tools: The control engineer uses block diagrams to specify feedback-controllers and the software engineer uses UML component diagrams and statecharts to specify the discrete coordination. In hybrid reconfiguration charts, both models are integrated to fulfill challenge (1). Challenges (2) and (4) are met by specifying deadlines for the transitions [6] and by the distinction between atomic and fading transitions.

Further, this approach enables advanced modeling of reconfiguration via multiple layers, as a change in the top-level component (Monitor) leads to an exchange of feedback-controllers which are multiple layers below the top-level component in the architectural view. The model and its visualization contain the separated configurations and a description when and how to switch between them. By the additional support of dynamic interfaces, our approach enables reconfiguration, based on local and external events (challenge (3)).

The visual integration of behavior and structure leads to models with reduced visual complexity: It simplifies analyses, as the reachable state space and the reachable configurations are visualized in the model: In the example, it is obvious that only the state combinations (*states of Monitor* $\times$ *states of Body Control*) (AllAvailable, Reference), (AbsAvailable, Absolute), (RefAvailable, Robust), and (NoneAvailable, Robust) are reachable. Others, e.g. (AllAvailable, Absolute), (AbsAvailable, Reference), cannot be reached. Inconsistent configurations, i.e. configurations that do not feed all required inputs, are detected without complex analyses. Furthermore, errors can be pinpointed to a single transition. As described in [8] in detail, a combination of transitions in different components guarantees consistent reconfiguration if the deadlines are not in conflict. For the intervals from our example, the consistency rule $d_3 \subseteq d_b$ must hold.

If we had used standard approaches, like the ones discussed in Section 3, we had to model the coordination between Monitor and Body Control by additional asynchronous communication, i.e. modeling the implied state changes by sending discrete signals. This would result in the same reachable state space, but it could not be derived intuitively and its determination would require additional effort.

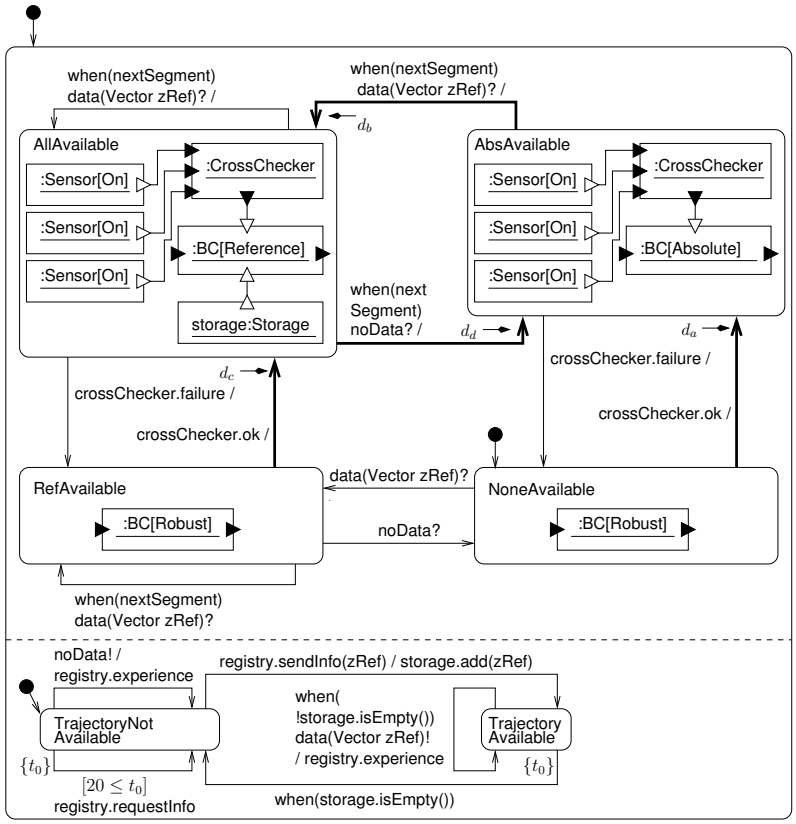

Figure 6. Behavior of the Monitor **component**

5. Flexible Structural Adaptation

The approach presented in the last section can effectively be applied when the required reconfiguration is local. Then usually all possible configurations are well-known at the design time and their number is small. However, specifying more flexible reconfiguration which results from the need to coordinate ad hoc groups cannot be addressed.

When, for example, shuttles build a convoy and a leader shuttle determines the reference positions for all the following shuttles, the control of these reference positions depends on the length of the convoy and on the participating shuttle types and characteristics. For example, a heavy load shuttle has to hold a larger distance within the convoy. The leader shuttle of a convoy can respect such individual properties or requirements only when individual components or feedback-controllers are applied to determine the reference positions. Using our approach, discussed in the last section, would thus be impractical as a large number of possible configurations (in principle even infinite many ones) have to be explicitly specified.

In the given example, the different shuttle types are not known a priori at design time (recall how many different types of automobiles exist). Thus, each shuttle sends the component, which the leader shuttle has to apply, to the leader shuttle when it joins the convoy at runtime. We therefore suggest to specify the required flexible structural reconfiguration by means of *reconfiguration rules* where control elements can be determined by parameters which are based on graph transformation systems (cf. [14]). Graph transformations are usually applied for model transformations (e.g. [23]). We will exemplify that they are even an appropriate visual, model-based description technique for the specification of reconfiguration at runtime and that the same advantages apply as for model transformations.

A cut-out of the behavior of the shuttles for *coordinating* convoys is depicted in Figure 7. Note that in [9] is described how to ensure a safe *building* of convoys. The hybrid reconfiguration chart consists of three states: ConvoyLeader represents that the shuttle is the leader shuttle, ConvoyFollower represents that the shuttle is part of a convoy but not the leader shuttle, and NoConvoy represents that the shuttle is not in a convoy at all.

Residing in state ConvoyFollower, the shuttle applies a position controller that delivers the current acceleration a dependent on its reference position s_{ref} and its current position $s_{current}$. It periodically receives the event receiveRefPos with parameter pos[] and stores the new reference po-

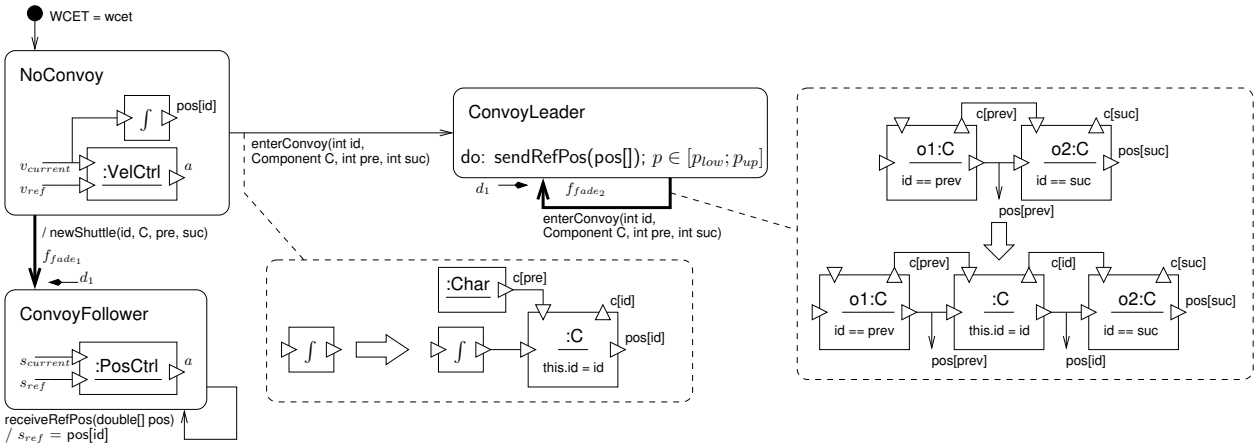

Figure 7. Shuttle behavior to control convoys

sition pos[id] as side-effect in s_{ref}. In state NoConvoy, the shuttle applies a velocity controller, requiring a reference and the current velocity as input. The latter one is used to determine the current position pos[id]. When a new shuttle joins the convoy, it sends an event enterConvoy with the following parameters: its identifier id, the component C to be used to determine the shuttle's reference position, and the IDs pre and suc of the shuttles which let the new shuttle in.

The reconfiguration rule of the transition (visualized with a dashed border) adds the component C to the shuttle's control structure: An instance of a component Char is created that provides the characteristics of the leader shuttle such as length, maximal brake acceleration etc. These characteristics and the current position of the leader shuttle are fed into C which determines the reference position pos[id] as output. A simple implementation of component C would just add the length of the preceding shuttle and an individual safety margin to the current position of the preceding shuttle. Port c[id] provides the characteristics of the new shuttle.

Residing in state ConvoyLeader, the shuttle sends periodically with a period $p \in [p_{low}; p_{up}]$ the reference positions pos[] to the according shuttles. If a further shuttle joins the convoy, its component is inserted in the structure between the components of prev and suc. Reconfiguration rules for the special cases when a shuttle joins at the end or at the beginning of the convoy are omitted in Figure 7. Due to lack of space, we omitted also transitions which model that shuttles leave the convoy. If we specified a transition, leading from ConvoyLeader to NoConvoy, the current configuration –eventually consisting of multiple components– would be discarded and the configuration of NoConvoy would be applied.

Especially rules which just create new components enlarge the execution time. In order to ensure predictability for the duration of the execution time which is indispensable for real-time systems, we define a worst case execution time wcet. The reconfiguration is just applicable when the execution time of the resulting structure is less or equal wcet. Only if the model is designed for simulation purposes, this can be neglected as the simulation times can be adjusted appropriately.

The example points out that modeling flexible reconfiguration with reconfiguration rules leads to an enormous reduction of the visual complexity, as not every possible configuration has to be specified explicitly.

6. Conclusion and Future Work

We presented the visual integration of object-oriented modeling techniques in form of the UML and control theory approaches in form of block diagrams within the MECHATRONIC UML approach. MECHATRONIC UML preserves where possible both the successful control engineering and software engineering concepts for modeling and additionally integrates them in an intuitive graphical manner. This integration enables us to describe mechatronic units which operate flexible and self-adaptive within a complex networked environment by adapting their structure at runtime to react to detected environmental changes, to change their system goals, or to react to a change of the system structure. In addition, a visual interface in form of the interface state charts permits the cooperation of experts from the different domains while they use the notations common in their domain.

The reconfiguration outlined in Section 4 has been prototypically realized in form of an integration of the open source CASE tool Fujaba[4] and the CAE tool CAMeL[5]. Planned future work includes to also realize the concepts

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

for the flexible reconfiguration in Fujaba on top of existing work for graph transformation systems (cf. [14]). Special emphasis has to be put here on appropriately implementing the real-time processing such that the approach is also applicable for hard real-time systems and on the verification of the reconfiguration rule based behavior. Currently, we evaluate the visual languages by realizing the presented examples in cooperation with control engineers.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(3-34), 1995.

[2] 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.

[3] K. Bender, M. Broy, I. Peter, A. Pretschner, and T. Stauner. Model based development of hybrid systems. In *Modelling, Analysis, and Design of Hybrid Systems*, volume 279 of *Lecture Notes on Control and Information Sciences*, pages 37–52. Springer Verlag, July 2002.

[4] K. Berkenkötter, S. Bisanz, U. Hannemann, and J. Peleska. Executable HybridUML and its Application to Train Control Systems. In H. Ehrig, W. Damm, J. Desel, M. Große-Rhode, W. Reif, E. Schnieder, and E. Westkämper, editors, *Integration of Software Specification Techniques for Applications in Engineering*, volume 3147 of *Lecture Notes in Computer Science*, pages 145–173. Springer Verlag, 2004.

[5] 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.

[6] S. Burmester, H. Giese, and M. Tichy. Model-Driven Development of Reconfigurable Mechatronic Systems with Mechatronic UML. In *Model Driven Architecture: Foundations and Applications*, volume 3599 of *Lecture Notes in Computer Science*, pages 47–61. Springer Verlag, 2005.

[7] V. Friesen, A. Nordwig, and M. Weber. Object-Oriented Specification of Hybrid Systems Using UMLh and ZimOO. In *Proceedings of the 11th International Conference of Z Users on The Z Formal Specification Notation, Berlin, Germany*, volume 1493 of *Lecture Notes in Computer Science (LNCS)*, pages 328–346. Springer Verlag, 1998.

[8] 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.

[9] 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.

[10] R. Grosu, T. Stauner, and M. Broy. A Modular Visual Model for Hybrid Systems. In *Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'98)*. Springer Verlag, 1998.

[11] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: The Next Generation. In *Proc. of the 16th IEEE Real-Time Symposium*. IEEE Computer Press, December 1995.

[12] Y. Kesten and A. Pnueli. Timed and Hybrid Statecharts and their Textual Representation. In J. Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of LNCS, Springer Verlag, 1992.

[13] C. Kobryn. Expertr's voice: UML 3.0 and the future of modeling. *Software and Systems Modeling*, 3(1):4 – 8, March 2004.

[14] H. J. Köhler, U. A. 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, Ireland*, pages 241–251. ACM Press, 2000.

[15] X. Liu, Y. Xiong, and E. A. Lee. The Ptolemy II Framework for Visual Languages. In *Proceedings of the IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC'01), Stresa, Italy*, pages 50–51, September 2001.

[16] N. Lynch, R. Segala, and F. 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 *Lecture Notes in Computer Science*, pages 403–417. Springer Verlag, 2001.

[17] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. Cheng. Composing Adaptive Software. *IEEE Computer*, 37(7), July 2004.

[18] P. J. Mosterman and G. Biswas. Modeling Discontinuous Behavior with Hybrid Bond Graphs. In *Proc. of the Intl. Conference on Qualitative Reasoning, Amsterdam, the Netherlands*, pages 139–147, May 1995.

[19] P. J. Mosterman and G. Biswas. A Theory of Discontinuities in Physical System Models. *Journal of the Franklin Institute*, 334B(6):401–439, January 1998.

[20] Object Management Group. *UML for System Engineering Request for Proposal, ad/03-03-41*, March 2003.

[21] Object Management Group. *Systems Modeling Language (SysML) Specification*, January 2005. Document ad/05-01-03.

[22] K. Ogata. *Modern Control Engineering*. Prentice Hall, 2002.

[23] G. Song, K. Zhang, and J. Kong. Model Management Through Graph Transformation. In *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC'04), Rome, Italy*, pages 75–82, September 2004.

[24] T. Stauner. *Systematic Development of Hybrid Systems*. PhD thesis, Technische Universität München, 2001.

[25] T. Stauner, A. Pretschner, and I. Péter. Approaching a Discrete-Continuous UML: Tool Support and Formalization. In *Proc. UML'2001 workshop on Practical UML-Based Rigorous Development Methods – Countering or Integrating the eXtremists*, pages 242–257, Toronto, Canada, October 2001.