

# Design and Simulation of Self-Optimizing Mechatronic Systems with Fujaba and CAMEL\*

Sven Burmester<sup>†</sup>, Holger Giese, and Florian Klein<sup>†</sup>  
Software Engineering Group, Warburger Str. 100, D-33098 Paderborn, Germany  
[burmi|hg|fklein]@upb.de

## ABSTRACT

Self-Optimizing mechatronic systems which are able to react autonomously and flexibly to changing environments are one promising approach for the next generation of mechanical engineering systems. To render designing such systems possible, an approach is required which goes far beyond what is offered by today's standard tools for mechatronic systems. In this paper, we outline how a smooth integration between mechanical and software engineering methods and tools supports the design of verifiable, complex, reconfigurable mechatronic systems. The focus of the paper is on enabling the design and simulation of safe reconfigurable mechatronic systems, as reconfiguration is a critical prerequisite for self-optimization.

## 1. INTRODUCTION

Mechatronic systems combine technologies from mechanical and electrical engineering as well as from computer science. In the future they will rather be composed of interacting systems than isolated solutions for individual devices. Networking and ever increasing local computational power enable sophisticated mechatronic systems, which, besides more advanced digital control, will include rather complex software coordination and information management capabilities. To handle the resulting complexity, each single unit of such composite systems must be able to react autonomously and flexibly to changing environmental settings.

To achieve the required flexibility, we propose to build self-optimizing technical systems which modify their goals endogenously based on changing environmental settings.<sup>1</sup> A critical prerequisite to realize a goal-compliant autonomous adaptation of the system behavior is the ability of the system to reconfigure its structure or parameters accordingly. This requires coordination between the mechanical engineering and software engineering elements of the system. Therefore a smooth integration between methods and tools from both domains is inevitable. The presented solution integrates the CASE tool Fujaba Real-Time Tool Suite<sup>2</sup> and the CAE tool

CAMEL<sup>3</sup> to support the design of verifiable, complex, reconfigurable mechatronic systems.

The paper is organized as follows: In Section 2, the semantic integration between block diagrams for mechatronic control systems and UML component models is summarized. Section 3 presents the necessary extensions to CAMEL (3.1) and Fujaba (3.2) and the binding tool and runtime environment required for ultimately integrating and executing the model (3.3). We then review relevant related work in Section 4 and present our final conclusions and give a short outlook on planned future work.

## 2. SEMANTIC INTEGRATION

As a running example, we will use the active vehicle suspension system for the shuttles from the *RailCab*<sup>4</sup> research project. In this project, a modular rail system will be developed; it is to combine modern chassis technology with the advantages of the linear drive technology (as applied in the Transrapid<sup>5</sup>) and the use of existing rail tracks. In our example, shuttle software is developed that realizes the safe switching between three different feedback controller structures, which control the body of the shuttle.

### 2.1 Control Engineering

Feedback controllers are usually specified through block diagrams or differential equations [11] and describe the relation between continuous in- and output signals. In our example, three different feedback controllers are applied, providing different levels of comfort to the passengers:

The controller *reference* uses the absolute acceleration of the coach body  $\ddot{z}_{abs}$  and a reference trajectory that describes the motion of the coach body  $z_{ref}$  as input signals. In case the trajectory is not available, the *absolute* controller, requiring only  $\ddot{z}_{abs}$ , has to be used. If neither the trajectory nor the measurement of  $\ddot{z}_{abs}$  are available, the *robust* controller is applied, requiring just common inputs (see Figure 1).

For switching between two controllers, one must distinguish two cases: When switching between the *normal* and the *failure* block in Figure 1, the change can take place between two computation steps (*atomic switching*). Switching between *reference* and *absolute* requires *cross-fading* in order to guarantee stability. The cross fading itself is specified by a fading function  $f_{switch}(t)$  and an additional parameter which determines the duration of the cross fading.

<sup>†</sup>Supported by the International Graduate School of Dynamic Intelligent Systems. University of Paderborn

\*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.

<sup>1</sup>[www.sfb614.de/eng/](http://www.sfb614.de/eng/)

<sup>2</sup>[www.fujaba.de](http://www.fujaba.de)

<sup>3</sup>[www.ixtronics.de](http://www.ixtronics.de)

<sup>4</sup><http://www-nbp.upb.de/en>

<sup>5</sup><http://www.transrapid.de/en>

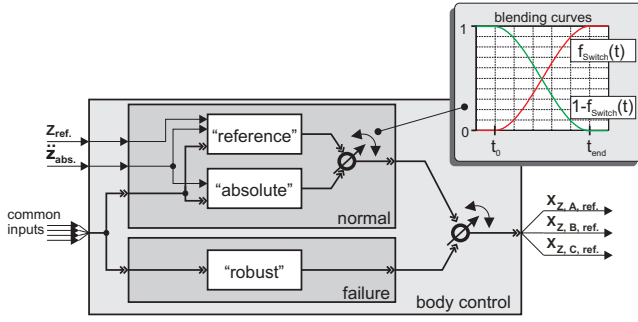


Figure 1: Different control modes and fading

## 2.2 Software Engineering

Inspired by ROOM [13], UML 2.0 supports the specification of the structure of complex systems using components with ports and deployment diagrams. The only support for real-time behavior is provided by the Profile for Schedulability, Performance, and Time [12]. In order to specify real-time behavior, we apply the real-time extension, the so called *Real-Time Statecharts* in Fujaba [2] as well as a restricted notion for Real-Time Patterns [5].

## 2.3 Integration

As proposed by the UML 2.0 approach, we use component diagrams to specify the overall structure of the system. We introduce *hybrid components* [4, 3], which integrate discrete and continuous behavior. To model communication of sporadic events and communication of continuously changing signals, we distinguish between discrete and continuous ports. The latter are visualized by a triangle inside the port-square, indicating the direction of the data flow.

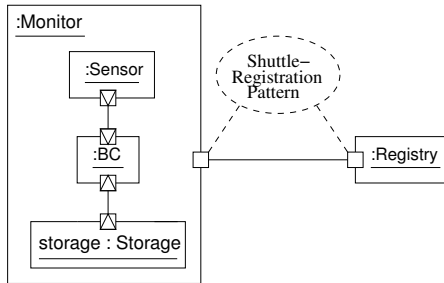


Figure 2: Monitor and its environment

In Figure 2, the structure of the shuttle's Monitor component is shown. It consists of the Sensor, delivering  $\ddot{z}_{abs}$ , the Body Control (BC) component, switching between the feedback controllers, and the Storage component used for storing the reference trajectory the Monitor obtains from a track section's Registry. A more detailed description can be found in [4, 3].

The Shuttle-Registration communication pattern in Figure 2 specifies the communication protocol between two components. Real-time model checking is used to verify the protocol. Compositional model checking and refinement relations enable even the verification of large, complex systems [5].

The behavior of hybrid components is specified using our notion of Hybrid Statecharts [4, 3], which extend the Real-Time Statecharts. In Hybrid Statecharts, each discrete state

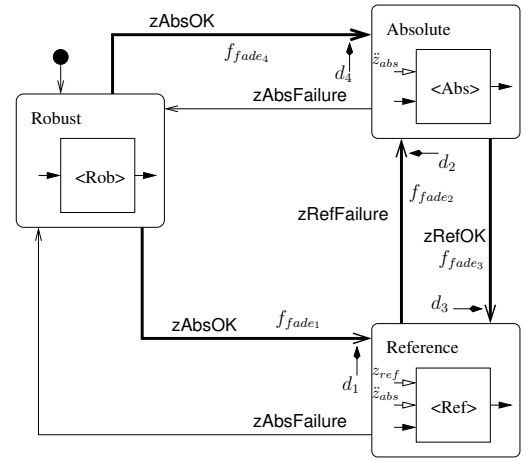


Figure 3: Behavior of the body control component

is associated with a configuration of embedded components. Figure 3 shows the behavior of the BC component as a simple example where each configuration consists of just one continuous feedback controller from Section 2.1.

State changes in Hybrid Statecharts are either modeled through *atomic* or *fading transitions*. The latter (visualized by thick arrows) can be associated with a fading function ( $f_{fade}$ ) and the required fading duration interval  $d = [d_{low}, d_{up}]$  specifying the minimum and maximum duration of the fading.

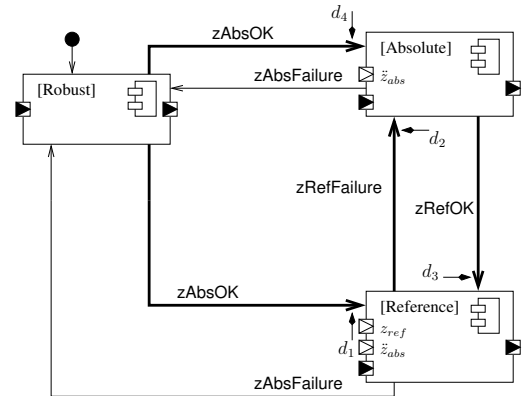


Figure 4: Interface Statechart of the BC component

As reconfiguration leads to changing interfaces (e.g. BC's continuous input signals are state-dependent), we provide the notion of hybrid *Interface Statecharts* (see Figure 4). They only consist of the externally relevant real-time information (discrete states, their continuous in- and outputs, possible state changes, their durations, signals to initiate transitions, and signal flow information [9]). They abstract from the embedded components and from the fading functions. Ports that are required in each of the three interfaces are filled in black.

Well-known approaches like hybrid automata [1] or HyCharts [6] embed only one continuous component, just as the Hybrid Statechart from Figure 3. This is insufficient, however, if reconfiguration is supposed to be possible at multiple levels, which requires hybrid components and their re-

configuration rather than merely the reconfiguration of the controllers.

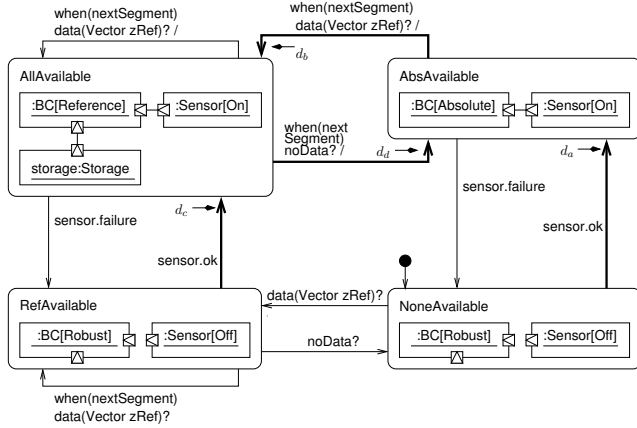


Figure 5: Monitor behavior with modular reconfiguration of the subcomponent BC

In our example, the Hybrid Statechart from Figure 5 specifies the behavior and the reconfiguration of the monitor component. It consists of four discrete states indicating which of the two signals  $\ddot{z}_{abs}$  and  $z_{ref}$  are available. Every discrete state has been associated with a configuration of the subcomponents BC, Sensor, and Storage.<sup>6</sup>

In the design of these configurations, only the interface description of the embedded components (e.g. Figure 4) is relevant as the inner structure can be neglected. Therefore, we specify the required structure and communication links for each discrete state and assign the BC component in the appropriate state to it. E.g., the BC component instance in state Reference has been assigned to the state AllAvailable where all signals are available. Therefore, a state switch in the Monitor statechart *implies* a state change in the BC statechart. Simple consistency checks ensure that the verified real-time behavior still holds in spite of embedding hybrid components [4].

### 3. TOOL INTEGRATION

Figure 6 illustrates the way these semantic concepts are used to achieve the desired tool integration between CAMEL and Fujaba: Both tools export hybrid components, which are then integrated into a common hierarchical model.

The tools' output is stored using an exchange format for the description of hybrid components. It contains a high-level interface description, consisting of the hybrid interface statechart (incl. signal flow information), a behavioral description at the source code level and a tool-specific section that allows subsequent modifications using the respective originating tool. As it is the de facto standard for mechatronic systems, C/C++ is used for the low level descriptions. The integration itself is carried out using only the interface descriptions, considering the individual components as black boxes.

#### 3.1 CAMEL

The CAE Tool CAMEL is used for modeling the dynamics of physical systems and for specifying feedback controllers.

<sup>6</sup>Note that the interaction with the Registry is not shown.

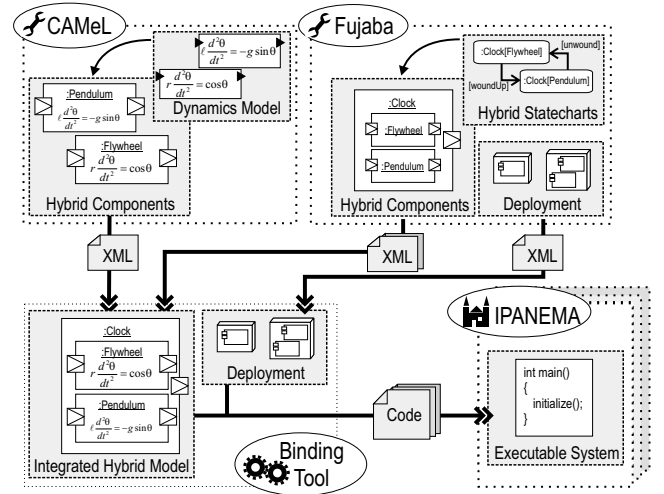


Figure 6: Tool Integration Overview

C++ code is generated from the designed block diagrams. It is executed or simulated within the run-time framework IPANEMA (see Section 3.3). In order to achieve the projected integration, controller block hierarchies may be exported as hybrid components, consisting of the required interface description and generated C++ code implementing the system's differential equations.<sup>7</sup> The required extensions are currently being implemented within the scope of a bachelor's thesis.

#### 3.2 Fujaba

Fujaba currently offers a wide range of UML-based diagrams for the complete specification of (real-time) software. Of particular interest in the current context are component diagrams, deployment diagrams, and Real-Time Statecharts. Discrete components already play a prominent role in the composition and verification of systems, and for reuse based on design patterns. From the specification, code for the Java Real-Time platform may then be generated.

Within the scope of a student research project, the tool suite is now being adapted to incorporate the proposed hybrid concepts by introducing support for hybrid components, ports, and statecharts. Building closely on the existing conceptual framework, the code generation is undergoing a massive rewrite in order to allow the generation of the required C++ code.

#### 3.3 Binding Tool & Run-Time Framework

Though the exchange format provides an integrated model of the complete component hierarchy, an additional step is required to carry this conceptual integration to the execution level. The binding tool determines the correct evaluation order from the signal flow information and then correctly interconnects the individual components.

As we do not consider the outlined integration approach as limited to Fujaba and CAMEL only, the binding tool (see Figure 6) under development operates at the interface specification level and merely binds the code generated by other tools without taking their internal model into account.

<sup>7</sup>Note that the interface statechart of a continuous component consists of just one discrete state.

This means that any tool can provide hybrid components for the binding tool, if it uses the exchange format and provides the C++ code itself. In order to ensure a correct integration, the generated code fragments need to adhere to the requirements set by a common run-time platform.

This platform is IPANEMA 2, a new run-time framework that will introduce reconfiguration and support for C++ into the existing C-based IPANEMA framework [7]. It aims to provide a common environment for the simulation of reconfigurable mechatronic systems, both in pure software and with hardware-in-the-loop (HIL) execution. Using the deployment information it receives from Fujaba, the binding tool is capable of setting up such a simulation and assigns the individual components to their respective nodes.

#### 4. RELATED WORK

Support for the integration of continuous behavior is currently not provided within standard UML. The need for such support is underlined by the OMG request for a proposal of UML for Systems Engineering [10].

In the hybrid extensions HyROOM [14] and HyCharts [6], two hybrid extensions of ROOM [13], complex architectures are specified in a way similar to ROOM, but the behavior is specified through statecharts whose states are associated with continuous models. Their approach, however, is restricted to a non-modular style of reconfiguration and does therefore, unlike the outlined approach, not support reconfiguration for complex, hierarchical systems.

A couple of modeling languages have been proposed to support the design of hybrid systems (e.g. [1, 15]). Most of these approaches provide models, like linear hybrid automata [1], that enable the use of efficient formal analysis methods but lack methods for structured, modular design and reconfiguration.

The de facto industry standard for modeling hybrid systems is MATLAB/Simulink and Stateflow.<sup>8</sup> Modeling reconfiguration is achieved by adding discrete blocks, whose behavior is specified by statecharts, to the block diagrams. Thus, continuous and discrete behavior are separated and not integrated as required for modeling the reconfiguration of complex systems.

#### 5. CONCLUSION AND FUTURE WORK

The presented approach is only a first step towards a complete integration between mechatronics and software engineering. It enables the seamless integration of CAE artifacts into UML in a modular fashion. Thus, the specific complexity and problems of the different disciplines such as the stability of the control behavior or the correct real-time coordination of the components can to some extent be addressed separately.

In the future, we plan to further strengthen and extend this integration. While currently the real-time processing and the quasi-continuous digital control are combined in a rather static manner, we plan to extend our approach to also cover more dynamic reconfiguration scenarios as well as compositional adaptation [8].

#### 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] S. Burmester and H. Giese. The Fujaba Real-Time Statechart PlugIn. In *Proc. of the Fujaba Days 2003, Kassel, Germany*, October 2003.
- [3] S. Burmester, H. Giese, and O. Oberschelp. Hybrid UML Components for the Design of Complex Self-optimizing Mechatronic Systems. In *Proc. of 1st International Conference on Informatics in Control, Automation and Robotics (ICINCO 2004), Setubal, Portugal*. IEEE, August 2004.
- [4] 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*. ACM, November 2004. (accepted).
- [5] 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.
- [6] 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), LNCS 1486*. Springer-Verlag, 1998.
- [7] U. Honekamp. *IPANEMA - Verteilte Echtzeit-Informationsverarbeitung in mechatronischen Systemen*. PhD thesis, University of Paderborn, 1998.
- [8] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. Cheng. Composing Adaptive Software. *IEEE Computer*, 37(7), July 2004.
- [9] O. Oberschelp, A. Gambuzza, S. Burmester, and H. Giese. Modular Generation and Simulation of Mechatronic Systems. In *Proc. of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI), Orlando, USA*, July 2004.
- [10] Object Management Group. *UML for System Engineering Request for Proposal, ad/03-03-41*, March 2003.
- [11] K. Ogata. *Modern Control Engineering*. Prentice Hall, 2002.
- [12] OMG. UML Profile for Schedulability, Performance, and Time Specification. OMG Document ptc/02-03-02, September 2002.
- [13] B. Selic, G. Gullekson, and P. Ward. *Real-Time Object-Oriented Modeling*. John Wiley and Sons, Inc., 1994.
- [14] 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.
- [15] R. Wieting. Hybrid high-level nets. In *Proceedings of the 1996 Winter Simulation Conference*, pages 848–855, Coronado, CA, USA, 1996.

<sup>8</sup>www.mathworks.com