Tool Support for Developing Advanced Mechatronic Systems: Integrating the Fujaba Real-Time Tool Suite with CAMeL-View*

Sven Burmester, Holger Giese,†
Stefan Henkler, Martin Hirsch,‡ Matthias Tichy
Software Engineering Group
University of Paderborn, Germany
{burmi,hg,shenkler,mahirsch,mtt}@uni-paderborn.de

Vadim Boiko, Alfonso Gambuzza, Eckehard Münch, Henner Vöcking
Control Engineering Group
University of Paderborn, Germany
{boiko,gamba,muench,hevoe}@rtm.uni-padernorn.de

Abstract

The next generation of advanced mechatronic systems is expected to use its software to exploit local and global networking capabilities to enhance their functionality and to adapt their local behavior when beneficial. Such systems will therefore include complex hard real-time coordination at the network level. This coordination is further reflected locally by complex reconfiguration in form of mode management and control algorithms. We present in this paper the integration of two tools which allow the integrated specification of real-time coordination and traditional control engineering specifically targeting the required complex reconfiguration of the local behavior.

1 Introduction

For mechatronic systems [2], which have to be developed in a joint effort by teams of mechanical engineers, electrical engineers, and software engineers, the advances in networking and processing power provide many opportunities. It is therefore expected that the next generation of advanced mechatronic systems will exploit these advances to realize more intelligent solutions where software is employed to exploit local and global networking capabilities to optimize and enhance their functionality by operating cooperatively. The cooperation in turn permits these systems to decide when to adapt their local behavior taking the information of cooperating subsystems into account.

The development of such advanced mechatronic systems will therefore at first require means to develop software for the complex hard real-time coordination of its subsystems at the network level. Secondly, software for the complex reconfiguration of the local behavior in form of mode management and control algorithms is required, which has to proper coordinate the local reconfiguration with the coordination at the network level.

The envisioned approach is complicated by the fact that classical engineers and software engineers employ different paradigms to describe their aspect of these systems. In software engineering discrete models such as state charts are frequently used to describe the required interaction, while the classical engineers employ continuous models to describe and analyze their control algorithms.

To enable the model-based development of the outlined advanced mechatronic system, an integration between these two paradigms is required which fulfills the outlined requirements. To provide an economically feasible solution, the required integration must further reuse the concepts, analysis techniques, and even tools of both involved paradigms where possible.

In [3], we demonstrated how to use the FUJABA REAL-TIME TOOL SUITE[1] to develop safety-critical real-time systems conform to the model-driven engineering (MDE) approach. We present in this paper the integration of two tools to bridge the development of software engineering for

*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.

†Currently a visiting professor at the Hasso-Plattner-Institut of the University of Potsdam.

‡Supported by the University of Paderborn

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

real-time systems with control engineering: the open source UML CASE tool FUJABA REAL-TIME TOOL SUITE and the CAE tool CAMeL-View. The employed concepts for the real-time coordination [9] and tool support for it have been presented in earlier work. For the the local reconfiguration only the concepts have been presented [6], while in this paper the developed tool support is described.

In the remainder of this paper, we first introduce the modeling concepts for the integrated description of discrete and continuous models in Section 2. Then, we outline in Section 3 how this conceptual integration has to be paralleled at the execution level. Afterward, we discuss the benefits of our approach with respect to analysis capabilities in Section 4 and compare our approach with the related work in Section 5. We finally provide our conclusions and an outlook on planned future work.

2 Integration at the model level

Modeling advanced mechatronic systems require the integration of modeling approaches used in software engineering and traditional engineering disciplines. To describe our approach for modeling these systems, we first introduce MECHATRONIC UML for specifying discrete parts of a system in Section 2.1. As mechatronic systems have continuous parts too, we introduce in Section 2.2 block diagrams. We finally provide our approach for modeling the required integration of the different modeling paradigms in Section 2.3.

2.1 Discrete Specification

The software architecture of the considered mechatronic systems is specified in MECHATRONIC UML [5] with components which are based on UML [11] components. The components are self-contained units, with ports as the interface for communication. A component can contain other components and events can be delegated from the top-level component to its subcomponents. The internals of a component are modeled with an extended version of UML State Machines. Ports are the only external access point for components and their provided and required interfaces specify all interactions which occur at the port. The interaction between the ports of the components takes place via connectors, which describe the communication channels.

As UML state machines are not sufficient to describe complex time-dependent behavior (cf. [10]), we introduced Real-Time Statecharts (RTSC) [4] as an appropriate modeling language for the discrete real-time behavior of a component and for the event-based real-time communication between components. Real-Time Statecharts contain various Timed Automata [1] related constructs. In contrast to Timed Automata, firing a transition in a RTSC consumes time.

2.2 Continuous Specification

Mechatronic systems contain software to continually control the mechanic behavior. The standard notation for control engineering is a block diagram which is used to specify feedback-controllers as well as the controlled plant. Consequently, our approach uses block diagrams for the structural and behavioral specification of continuous components.

Block diagrams generally consist of basic blocks, specifying behavior and hierarchy blocks that group basic and other hierarchy blocks. Each block has input and output signals. The unidirectional interconnections between the blocks describe the transfer of information. The behavior of basic blocks is usually specified by differential equations, specifying the relationship between the block's inputs and outputs.

2.3 Hybrid Specification: Integration of Feedback-Controller Configurations

As mentioned in Section 1, for mechatronic systems, it is not sufficient to specify how discrete states of a component change: Dependent on the current state, the components have to apply different feedback-controllers. In [5], we introduced a new approach for the specification of hybrid behavior, which integrates discrete and continuous behavior, and even supports reconfiguration.

The idea is to associate to each discrete state of a component a configuration of subordinated components. Such a configuration consists of the subordinated components and their current connection. These components are either pure continuous components (feeback-controllers) or discrete or hybrid components. If the subordinated components are discrete or hybrid components, the configuration of these subordinated components consists also of the current state of the subordinated discrete or hybrid component. As hybrid components have a dynamic interface and shows just the ports required in their current state, also a configuration of components show just the in- and out-ports which are really required or used in the current state of the superordinated component.

This kind of modeling leads to implied state changes: When the superordinated component changes its state, this implies reconfiguration of the subordinated components. The subordinated components reconfigure their communication connections and –if specified– their discrete state. Such implied state changes can imply further state changes, if the subordinated components embed further components. This kind of modeling leads to reconfiguration across multiple hierarchical levels.

Compared to state of the art approaches, this approach has the advantage that models are of reduced size and that analyses require reduced effort (see Section 4).

3 Integration at runtime

Our approach for developing reconfigurable mechatronic systems applies the model-driven development approach to develop software systems at a high level of abstractions to enable analysis approaches like model checking. Therefore, ideally, we start with platform independent models to enable the compositional formal verification (cf. [9]). Afterward, the platform independent model must be enhanced with platform specific information to enable code generation. The required platform specific information is based on a platform model, which specifies the platform specific worst case execution times. After providing the platform specific information, we generate code from the models. In the remainder of this section, the generated code is described. Therefore, we introduce the evaluation of the system's components. Due to continuous and discrete parts of the considered mechatronic systems we have to consider data flow and event based evaluation. Complex reconfigurations lead to a lot of possible evaluation configurations and requires synchronization between the discrete and continuous parts. Our approach assures reconfigurations by the evaluation of the data flow and further we assure the data flow despite of reconfiguration. In the next subsections (Section 3.1, Section 3.2, and Section 3.3), we introduce our approach by considering first the discrete evaluation, then the continuous evaluation, and finally the hybrid evaluation.

3.1 Discrete Evaluation

When a component is evaluated, it triggers periodically the evaluation of its embedded components. As not every embedded component belongs to every configuration, it depends on the current discrete state of the component which of the embedded components are evaluated. Then the triggered components will themselves trigger their embedded components (in dependency of their discrete states) and so forth. Further, the association of configurations to discrete states leads to reconfiguration when a component changes its discrete state (e.g. when receiving an event). Due to the implied state changes (see Section 2.3), this leads to reconfiguration of the embedded components which are pure discrete, pure continuous or hybrid components.

3.2 Continuous Evaluation

The continuous parts of a configuration describe the data flow between the continuous inputs and the continuous outputs of the system. To ensure stability of the continuous part of the system, the data flow may not be interrupted within a computation step. Consequential, reconfiguration may only occur between two computation steps. Therefore, we separate execution of the continuous, data flow-orientated, and the discrete, event-based, parts: At the beginning of a period, the continuous system parts are evaluated. This is followed by the evaluation of the discrete system parts. Thus, we ensure that the reconfiguration –which takes place in the discrete part– occurs after the computation of the continuous system part is finished.

3.3 Hybrid Evaluation

Besides separating the continuous system parts from the discrete ones, it has to be managed which components need to be evaluated in which discrete state. Enhancing the top-level component with this information is usually not feasible as the number of global states grows exponentially with the number of components. Therefore, we compose the whole system as a tree structure consisting of single Hybrid Components to obtain an efficient implementation. Each Hybrid Component contains the information about its discrete and continuous parts –which may consist of system parts of the embedded components– itself. By the presented integration at runtime, we ensure consistent, correct data flow and an efficient implementation in spite of complex reconfiguration. Our seamless approach is realized by the Fujaba Real-Time Tool Suite in combination with the CASE Tool CAMeL. Both tools export hybrid components for the integration on the modeling level (see Section 2.3) and they export C++ code which is integrated to realize the hybrid, reconfiguration behavior.

4 Analysis capabilities

For the outlined MECHATRONIC UML approach, two specific verification tasks for the resulting systems are supported.

First, the MECHATRONIC UML approach supports model checking techniques for real-time processing at the network level. It addresses the scalability problem by supporting a compositional proceeding for modeling and verification exploiting the component model and the corresponding definition of ports and connectors as well as patterns [9].

Secondly, a restricted subset of the outlined hierarchical component structures for modeling of discrete and continuous control behavior can be checked for the consistent reconfiguration and real-time synchronization w.r.t reconfiguration taking proactive behavior into account [6, 8].

As the second approach can be embedded into the first one, a combination of both approaches cover the whole real-time coordination issues from the network level down to the reconfiguration of the lowest level components.

5 Related Work

Related tools and techniques to MECHATRONIC UML are CHARON, Hybrid UML with HL^3, HyROOM/HyCharts/Hybrid Sequence Charts, Massacio and Giotto, Matlab/Simulink/Stateflow, Ptolemy II, and UML^h [7].

All presented tools and techniques support the specification of a system's architecture or structure by a notion of classes or component diagrams. All approaches support modular architecture and interface descriptions of the modules. Nevertheless, they do not respect that a module can change its interface due to reconfiguration which can lead to incorrect configurations.

CHARON, Masaccio, HybridUML with HL^3, UML^h, HyROOM, and HyCharts have a formally defined semantics, but due to the assumption of zero-execution times or zero-reaction times, most of them are not implementable, as it is not realizable to perform a state change infinitely fast on real physical machines. CHARON is the only approach providing an implementable semantics. HyCharts are implementable after defining relaxations to the temporal specifications. They respect that idealized continuous behavior is not implementable on discrete computer systems. Further, CHARON provides a semantic definition of refinement which enables model checking in principle. Ptolemy II even provides multiple semantics and supports their integration.

Although most of these approaches enable ruling the complexity by a modular, component-based architecture and by behavioral models that support history and hierarchical and orthogonal states, reconfiguration across multiple hierarchical levels as required for the advanced mechatronic systems envisioned and provided by the presented approach is supported by none of them.

6 Conclusion

The tool integration of the CAE Tool CAMeL-View and the CASE Tool Fujaba Real-Time Tool Suite enables the application of our approach by continuing using well-approved tools. It does not only integrate models, but also the synthesized source code.

Acknowledgments

We thank all students who helped building the FUJABA REAL-TIME TOOL SUITE within student research projects, master, and bachelor theses, namely Vadim Boiko, Margarete Kudak, Wladimir Pauls, Matthias Schwarz, Björn Schwerdtfeger, and Andreas Seibel.

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for Real-Time Systems. In *Proc. of Logic in Computer Science*, pages 414–425. IEEE Computer Press, June 1990.

[2] D. Bradley, D. Seward, D. Dawson, and S. Burge. *Mechatronics*. Stanley Thornes, 2000.

[3] S. Burmester, H. Giese, M. Hirsch, D. Schilling, and M. Tichy. The Fujaba Real-Time Tool Suite: Model-Driven Development of Safety-Critical, Real-Time Systems. In *Proc. of the 27th International Conference on Software Engineering (ICSE), St. Louis, Missouri, USA*, pages 670–671, May 2005.

[4] S. Burmester, H. Giese, and W. Schäfer. Model-Driven Architecture for Hard Real-Time Systems: From Platform Independent Models to Code. In *Proc. of the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA'05), Nürnberg, Germany*, Lecture Notes in Computer Science, pages 25–40. Springer Verlag, November 2005.

[5] S. Burmester, H. Giese, and M. Tichy. Model-Driven Development of Reconfigurable Mechatronic Systems with Mechatronic UML. In *Model Driven Architecture: Foundations and Applications*, LNCS 3599, pages 47–61. Springer-Verlag, August 2005.

[6] 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, November 2004.

[7] 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.

[8] H. Giese and M. Hirsch. Modular Verificaton of Safe Online-Reconfiguration for Proactive Components in Mechatronic UML. In J.-M. Bruel, editor, *Satellite Events at the MoDELS 2005 Conference, Montego Bay, Jamaica, October 2-7, 2005, Revised Selected Papers*, LNCS 3844, pages 67–78. Springer Verlag, January 2006.

[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] S. Graf and I. Ober. A Real-Time profile for UML and how to adapt it to SDL. In *Proceedings of the SDL Forum'03*, 2003.

[11] Object Management Group. *UML 2.0 Superstructure Specification*, October 2004. Document: ptc/04-10-02 (convenience document).