$FRITS^{Cab}$: Fujaba Re-Engineering Tool Suite for Mechatronic Systems*

Stefan Henkler, Moritz Breit, Christopher Brink, Markus Böger, Christian Brenner, Kathrin Bröker, Uwe Pohlmann, Manel Richtermeier, Julian Suck, Oleg Travkin, Claudia Priesterjahn
Software Engineering Group
University of Paderborn
Warburger Str. 100
D-33098 Paderborn, Germany
[shenkler,mbreit,cbrink,markusb,cbr,kathyb,upohl,mane02,jsuck,oleg82,cpr]@uni-paderborn.de

ABSTRACT

Mechatronic systems use their software to enable enhanced functionalities. Due to the complexity of these systems model-driven engineering of the software has become the means to construct reliable software. As safety is of paramount importance for these systems, legacy components, which have shown their quality in practice, are often reused and adjusted to new requirements. Therefore, the re-engineering of legacy components is important. We present an approach for the re-engineering of mechatronic systems which focuses especially on distributed real-time and safety requirements and an integration into a model-driven engineering approach.

1. INTRODUCTION

Mechatronic systems, like automotive systems and aerospace systems, are usually networks of mechatronics components. Software is used to enable communication and thus to exploit knowledge of other components to enhance the functionality and to adapt the behavior of a single component when beneficial. Adapting the behavior might require complex reconfigurations of controllers in the form of mode management and control algorithms under hard realtime constraints.

Due to the complex nature of networked mechatronic systems and their usually safety-critical operations, i.e. lives may be at risks in case of failure, model-driven engineering of the software has become the means to construct reliable software as this is an enabler for a model-based analysis.

As safety is of paramount importance for mechatronic systems, legacy components, which have shown their quality in practice, are often reused. Therefore, the integration of legacy components into a model-driven engineering approach and the adjustment to new requirements is important.

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

The overwhelming complexity of the interaction of distributed real-time components usually excludes that testing alone can provide the required coverage when integrating a legacy component. Thus formal verification techniques seem to be a valuable alternative. However, the required verification of the resulting system often becomes intractable as no abstract model of the reused components is available, which can serve the verification purpose.

In this paper, we present an approach for the re-engineering of mechatronic systems by extending the MECHATRONIC UML approach. Current approaches in the re-engineering community do not consider mechatronic systems (see related work section). While the general question is the same , the characteristics of mechatronic systems require other approaches that focus especially on distributed real-time and safety requirements and an integration into a model-driven engineering approach. The following aspects are addressed: 1) the integration of legacy components into a model-driven engineering approach (MECHATRONIC UML) by taking safety requirements into account, 2) a self-adaptive engineering approach for mechatronic systems to enable the application of these systems in dynamic and changing environments in order to reduce the costs for maintenance. We present only a sketch of the approach. For more details we refer to [MB09]. The approach is implemented in the Fujaba Real-Time Tool Suite. Figure 1 gives a short overview of the development process of $FRITS^{Cab}$.

In the next section, we present the integration of legacy components into a MECHATRONIC UML based component architecture by taking safety requirements into account. We show how we can learn the relevant behavior information for the integration of a legacy component by proving that specified constraints for the integration are fulfilled (the legacy component is a refinement of the expected behavior). We extend our previous work [GHH08] by taking purely black-box-, white-box- legacy components, and controller behavior (which is of importance for mechatronic systems) into account. To address the problem of the increasing maintenance costs for systems in dynamic and changing environments, we describe a self-adaptive engineering approach for mechatronic systems in Section 3. We especially focus on

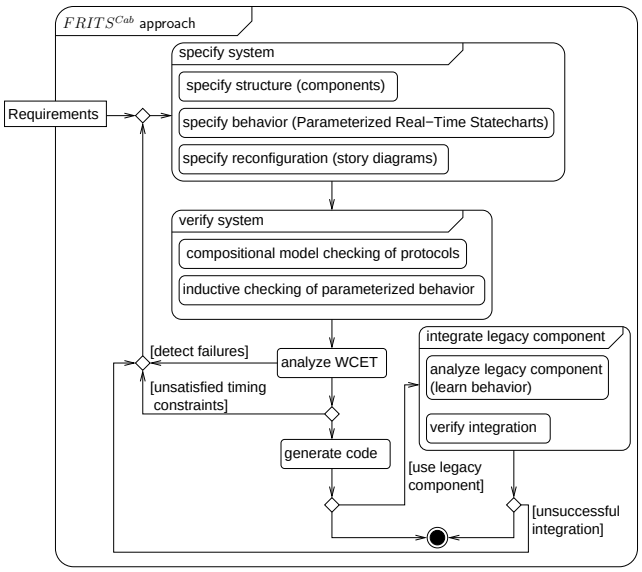

Figure 1: Overview of $FRITS^{Cab}$

worst-case-execution-time (WCET) analysis and code generation. For more details on modeling and (formal) verification, we refer to [GHH08]. After reviewing the relevant related work in Section 4, we conclude with a summary and future work.

2. INTEGRATION OF LEGACY COMPONENTS

Reverse engineering [CI90] is the discipline which takes the analysis and understanding of (legacy) software systems into account. As reverse engineering is a time intensive task, (semi-) automatic approaches are required for analyzing the relevant information. In our case, an approach is required which can learn the relevant behavior information for integrating a legacy component by proving that the specified constraints for the integration are fulfilled.

As explained in the introduction, we can differ between white-box legacy components (the source code and all relevant information for the execution of the legacy component are known) and black-box legacy components (only the relevant information of the interface and the execution of the legacy component are known). In the following, we first describe an approach for taking white-box legacy components into account. Afterwards, we consider black-box legacy components and the integration of system identification to enable the reverse engineering of controllers. We will close this section with a short discussion.

Figure 2 gives an overview of the integration approach. The integration for the white-box- and black-box- approach can either show a refinement of the abstract role, if specified, or can show a correct communication with the communicating (modeled) component. Then, based on the learned state behavior the controller behavior is learned by system identification. Hence, it is possible to identify a reconfiguration of controller behavior. Figure 3 shows a component architecture including a legacy component, which is analyzed in the following.

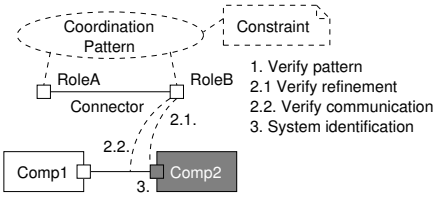

Figure 2: Overview of integration

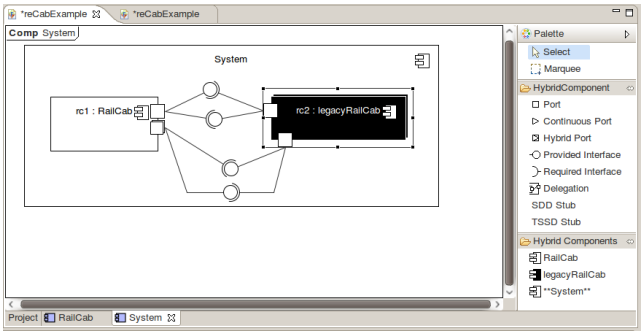

Figure 3: (Legacy-) Component Architecture

White-Box. A precondition for this approach is, that the source code (in our case C) of the legacy component is available. Furthermore, we have to know the interface procedures which are responsible for the communication. The basic idea is to use a source code model checker like BLAST[1] or CBMC[2] to enable a compositional formal verification or refinement on the source code level. By refinement or formal verification of the communication behavior, the approach supports as input a C code legacy component and a C code component of the communicating modeled component or the C code of the abstract modeled role (of the legacy component).

In a first step, we have to map the (abstract) model to source code to enable the required check (with the legacy component). The mapping has to preserve the execution semantics of the model. Hence, one possible (deterministic) path of the model has to be mapped to the source code.

As the considered kind of systems are reactive once the generated system is executed periodically (that is also the case for the legacy component). The WCET of a transition has to be within the specified deadlines (the WCET analysis is explained in Section 3). Within a period a task can be executed undeterministically by the scheduler. Furthermore, the periods of the legacy system and the model can be different.

Our Tool Suite supports a C code generation which considers the discussed requirements. Based on the generated code and a runtime framework which encapsules the communication calls as well as a simulated time we can start the proof

[1]http://mtc.epfl.ch/software-tools/blast/
[2]http://www.cprover.org/cbmc/

with a source code model checker. Conceptually the BLAST model checker fits the best as this model checker supports a good abstraction based on lazy abstraction. But, in our evaluation BLAST had a lot of problems with bounded arrays which are required for the communication. The bounded model checker CBMC supports the most C constructs and supports also the required bounded arrays. Hence, we use CBMC for our evaluation (Figure 4 shows the triggering of the white-box checking and Figure 5 gives an overview of the parameters of the white-box checking). Figure 6 shows a part of the behavior which we have verified on the code level. The used model checker does only scale up for small examples. The cause is not the number of states but rather the number of non-determinism due to a correct mapping of the scheduling and for timing.

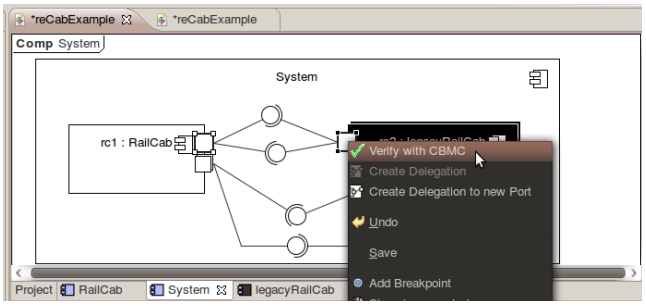

Figure 4: Tooling of White-Box Checking

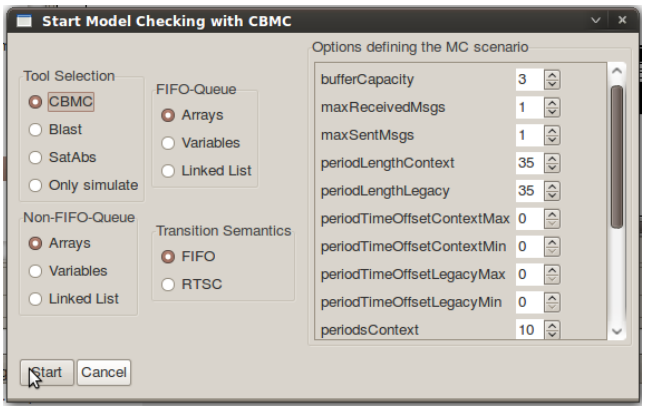

Figure 5: Overview White-Box Checking

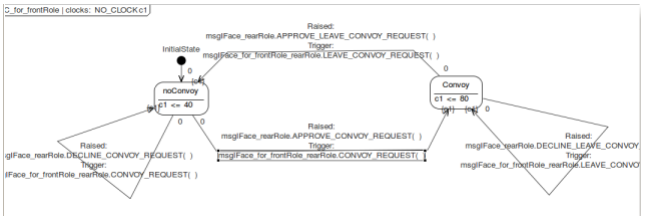

Figure 6: Verified behavior

Black-Box. A precondition for the black-box integration are the interface procedures which are responsible for the communication and all other relevant information for the execution of the legacy component (e. g. like the period). We extend the existing work of [Ang87] by considering the context in form of a possible direct input for the membership queries[3] by verifying the learned behavior with the modeled behavior after each learning step. If the counter example is also true on the code level the integration is unsuccessful. Otherwise, if the learned behavior has been determined[4] and the model checking is successful, the integration is correct if the number of states are an upper bound of the state number of the legacy component. Because of the known counter part of the legacy component, we can determine the number of states by the number of states of the modeled behavior. In most cases this should be a good approximation as in the considered domain with hard real-time requirements, typically every protocol consists of a watchdog pattern. That means, after each call an acknowledgment within a specified time is required which is an indication of a state. The implemented concepts (including a caching for the membership queries) yield fast evaluation results. We have evaluated our approach by the presented behavior in Figure 6. Furthermore, we tested a lot of other behavior up to 15 states. The time for learning the correct behavior for 15 states is about two minutes. Figure 7 shows the tool integration of the black-box approach.

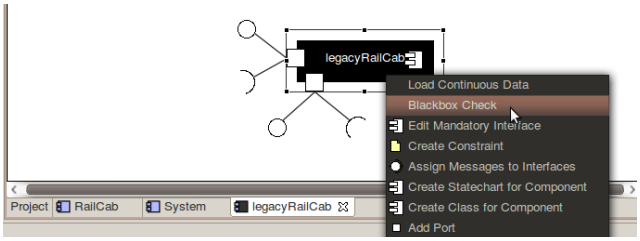

Figure 7: Black-Box Checking

System Identification. Before identifying the controller behavior, the state behavior has to be known. Hence, we require that the learning of the state behavior described above is successful. This approach is not applicable with the white-box approach as the applied tools do not support an externally visible model of the checked systems (accordingly, an extension of the CBMC or Blast tool is future work)[5]. As described in the introduction, besides the communication also the different (controller) modes are of importance for a mechatronic system. System identification [FPW98], is the approach which enables the identification of a controller algorithm. This is done by simulation. We can simulate each path of the learned behavior and for each state we can identify the controller behavior. The input of the system identification is a specified test trajectory or a realistic run with its environment. Based on the input and output behavior the transfer function of the controller is identified for linear systems. If the transfer functions are known, we can identify reconfigurations by different transfer functions.

[3]This kind of queries are used to learn the behavior
[4]That is the case if the equivalence query is successful
[5]Constraints of the controller behavior can be supported by the tools.

This approach supports the engineer in integrating mechatronic components into early development phases. Typically the engineers test the legacy components (controllers) only in hardware-in-the-loop scenarios or the real environment later in the development process. In order to get realistic simulation runs we can use our deterministic framework [GH06]. In $FRITS^{Cab}$ we have integrated the MATLAB System Identification Toolbox[6] (see Figure 8). This enables the user to identify the controller behavior of each state.

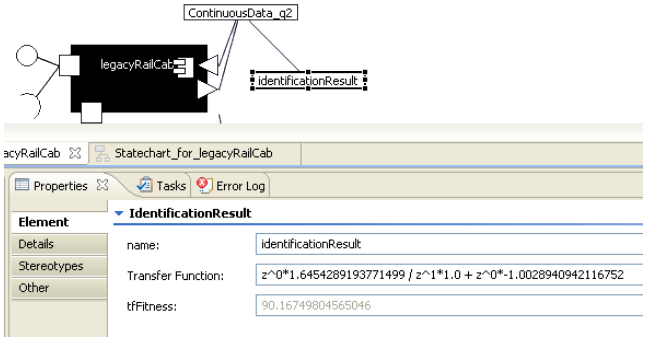

Figure 8: Integration of System Identification

Discussion. The approach therefore extends our former work and has the benefit that it could pin-point real failures in the test step which are no false negatives right from the beginning. In addition, it can also prove the correctness of the integration for an abstract behavioral model of the legacy component without learning the whole legacy component by only checking possible integration problems for the explicit given context. This is true for the white-box case. For the black-box approach we require an upper number of the states of the relevant behavior for the integration of the black-box component. The presented integrated system identification approach enables to also identify the controller behavior. Together with the learned state behavior, we can identify a restricted class of hybrid systems which change the configuration (the controller behavior) event triggered. For the complete class of hybrid systems system identification is not possible as a reconfiguration can be triggered at any time by a continuous value [Lju08]. In our former work, we have shown that the considered class of hybrid systems are of relevance for mechatronic systems [GBSO04]. In this section, we present only a one port to one port integration. If the legacy component has a dependency to more than one port the parallel product of the dependent ports is used.

3. FORWARD ENGINEERING

Mechatronic systems often require a dynamic structure and an adaptation to a changing environment. Typically, legacy components do not consider these requirements. In the next section we describe a specification approach which takes dynamic structures and a changing environment into account in order to reduce maintenance costs.

[6]http://www.mathworks.com/products/sysid/

Dynamic Structural Specification. There are a lot of examples in the context of network mechatronic systems, which require a dynamic structure and an adaptation to the changing environment. An often used example is a convoy of autonomous vehicles for reducing the energy or for extending the comfort of a driver (of a car) or for driving cars through a road work in a safe manner[7].

As oftentimes the collaboration between a flexible number of participants is required in these examples, we extend in [HHG08] the 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 with an open number of participants. To enable this kind of specification we extend the classical state based specification of behavior by Story Diagrams. The typical approach in the literature for modeling systems with hard real-time requirements is to give an upper limit for this kind of dynamics. This is a cause for a system not running appropriately as the upper limit is only an estimate and therefore a cause for reworking or adapting the system.

WCET Analysis and Code Generation. Besides the challenge of specifying and analyzing dynamics in an appropriate manner (we refer to [HHG08] for more details), we have to analyze the WCET and generate code. For both cases, in this paper, we especially consider the reconfiguration. The main problem of both parts is that the possible evolution of the system is not known a priori. This means, that a restriction of the evolution is not adequate by restricting for example the number members in a convoy. In principle, for a hard real-time system it is important to guarantee the worst case execution time and memory consumption. As presented in [THHO08] we can avoid this problem by a service level scheduling approach which takes application information into account. Hence, we specify profiles which provide information for the scheduler (for simplification we take in this paper only the WCET into account). The profile specifies an upper bound for the number of instances the application can use. During runtime, the application can request more resources if required and adjust the profile. The service level scheduler will support the request, if another profile can be reduced. Hence, the approach will work well, if for each configuration a few profiles are specified. Our code generation supports this approach by a factory implementation for the instances. The upper bound of the factory is that of the profile. During runtime we can adjust this bound dynamically. Furthermore, we guarantee that methods, in our case implemented with Story Diagrams, do not harm the instance bound. To enable this approach, we analyze the WCET for methods a priori (which are in our case implemented by story diagrams, see Figure 9). As the matching ordering is guaranteed by the code generation, we can generate a formula for each story diagram which we can use during runtime to compute the exact WCET for each new profile or adjustment of a profile. As we also know the worst case computation time of the formula, we can support the dynamics as well as a safe scheduling.

[7]Our test bed for proving such examples is the RailCab project (http://www-nbp.uni-paderborn.de/).

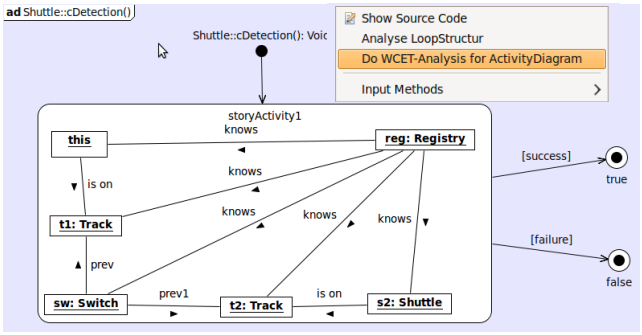

Figure 9: Static WCET analysis

4. RELATED WORK

Current re-engineering approaches for a component based system [CKK08] consider how to identify a (composite) component architecture. [Hus07] considers the reverse engineering of real-time systems by recording traces by instrumenting the platform. Impact of changes on the behavior of real-time systems are presented in [And05]. These approaches are based on simulation and platform specific instrumentation. [Jen01] consider learning and checking of Uppaal models. Black box and adaptive model checking based on learning are presented in [GPY02]. Some of these approaches do not consider real-time and safety requirements. All approaches do not consider a safe integration and an early verification of safety properties.

Several approaches and tools for WCET estimation, like aiT from AbsInt[8] or Bound-T from Tidorum already exist. Most of them handle low-level analysis to estimate the WCET of executable code [WRKP05]. There also exist a number of tools for code generation. None of them considers dynamic structures and time.

5. CONCLUSION AND FUTURE WORK

In this paper, we presented $FRITS^{Cab}$, a tool for the re-engineering of mechatronic systems. The tool is integrated into the Fujaba Real-Time Tool Suite. $FRITS^{Cab}$ enables the specification, analysis and code generation of mechatronic systems. In this paper, we especially focus on the WCET analysis and code generation of this kind of systems. The tool also supports the integration of legacy components by taking safety requirements into account. Currently, we especially focus on the evaluation of the integration approach by industrial applications.

6. REFERENCES

[And05] Johan Andersson. Modeling the temporal behavior of complex embedded systems - a reverse engineering approach. Licentiate thesis, June 2005.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[CI90] Elliot J. Chikofsky and James H. Cross II. Reverse engineering and design recovery: A taxonomy. *IEEE Softw.*, 7(1):13–17, 1990.

[CKK08] Landry Chouambe, Benjamin Klatt, and Klaus Krogmann. Reverse engineering software-models of component-based systems. In *12th European Conference on Software Maintenance and Reengineering, CSMR 2008, April 1-4, 2008, Athens, Greece*, pages 93–102, 2008.

[FPW98] G. F. Franklin, J. D. Powell, and M. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley Longman Publishing Co., Inc., 3 edition, 1998.

[GBSO04] Holger Giese, Sven Burmester, Wilhelm Schäfer, and Oliver Oberschelp. Modular Design and Verification of Component-Based Mechatronic Systems with Online-Reconfiguration. In *Proc. of 12th ACM SIGSOFT Foundations of Software Engineering 2004 (FSE 2004), Newport Beach, USA*, pages 179–188. ACM Press, November 2004.

[GH06] Holger Giese and Stefan Henkler. Architecture-driven platform independent deterministic replay for distributed hard real-time systems. In *Proceedings of the 2nd International Workshop on The Role of Software Architecture for Testing and Analysis (ROSATEA2006)*, pages 28–38, New York, NY, USA, July 2006. ACM Press.

[GHH08] Holger Giese, Stefan Henkler, and Martin Hirsch. Combining Compositional Formal Verification and Testing for Correct Legacy Component Integration in Mechatronic UML. In Rogério de Lemos, Felicita Di Giandomenico, Cristina Gacek, Henry Muccini, and Marlon Vieira, editors, *Architecting Dependable Systems V*, volume 5135 of *LNCS*, pages 248–272. SPRINGER, 2008.

[GPY02] Alex Groce, Doron Peled, and Mihalis Yannakakis. Adaptive model checking. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume Volume 2280/2002, pages 269–301. Springer Berlin / Heidelberg, 2002.

[HHG08] Martin Hirsch, Stefan Henkler, and Holger Giese. Modeling collaborations with dynamic structural adaptation in mechatronic uml. In *SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pages 33–40, New York, NY, USA, 2008. ACM.

[Hus07] Joel Huselius. *Reverse Engineering of Legacy Real-Time Systems: An Automated Approach Based on Execution-Time Recording*. PhD thesis, Mälardalen University, Computer Engineering, 2007.

[Jen01] Peter Krogsgaard Jensen. *Reliable real-time applications - and how to use tests to model and understand*. PhD thesis, Aalborg University, 2001.

[Lju08] Lennart Ljung. Perspectives on system identification. In *Proc. 17th IFAC World Congress*, Seoul, Korea, July 2008. (Plenary session).

[MB09] Christian Brenner Kathrin Bröker Uwe Pohlmann Manel Richtermeier Julian Suck Oleg Travkin Moritz Breit, Markus Böger. *Abschlussarbeit der Projektgruppe ReCab: Re-Engineering mechatronischer Systeme*, 2009. to appear.

[THHO08] Matthias Tichy, Stefan Henkler, Jörg Holtmann, and Simon Oberthür. Towards a Transformation Language for Component Structures. In *Postproc. of the 4th Workshop on Object-oriented Modeling of Embedded Real-Time Systems (OMER 4), Paderborn, Germany*, pages 27–39, 2008.

[WRKP05] Ingomar Wenzel, Bernhard Rieder, Raimund Kirner, and Peter Puschner. Automatic Timing Model Generation by CFG Partitioning and Model Checking. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 606–611, Washington, DC, USA, 2005. IEEE Computer Society.

[8] http://www.absint.com/ait