

# The Fujaba Real-Time Tool Suite\*

## Model-Driven Development of Safety-Critical, Real-Time Systems

Sven Burmester<sup>†</sup>, Holger Giese, Martin Hirsch<sup>‡</sup>, Daniela Schilling<sup>†</sup>, Matthias Tichy  
Software Engineering Group, University of Paderborn, D - 33095 Paderborn, Germany

[burmi|hg|mahirsch|das|mtt]@uni-paderborn.de

### Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques; D.2.4 [Software Engineering]: Software/Program Verification; D.2.13 [Software Engineering]: Reusable Software

### General Terms

Design, Verification

### Keywords

UML, Safety-Critical, Real-Time, Embedded Systems, MDA, Model Checking

## 1. INTRODUCTION

More and more complex functionality is today realized with complex, networked, real-time systems. The majority of the costs and time in development is required to design and verify the control software. As these systems are often used in a safety-critical environment, the FUJABA REAL-TIME TOOL SUITE aims at supporting the model-driven development of correct software for such safety-critical, networked, real-time systems. A restricted high level UML model serves as a basis for rigorous verification by means of model checking. Our incremental model checking is integrated with the usually incremental and iterative design. The state explosion problem is reduced by our compositional reasoning approach [3]. Analyzing the high level models rather than the code is justified by synthesis techniques, which preserve the real-time behavior of the model.

## 2. MODEL-DRIVEN DEVELOPMENT

To address the complexity of the system design at the architectural level, we propose to use a subset of the current UML 2.0 proposal (cf. [2]). The presented tool supports the modeling of the system structure by means of UML component diagrams and the modeling of real-time behavior by

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

<sup>‡</sup>Supported by the University of Paderborn

means of real-time extended UML state machines named Real-Time Statecharts (RTSC).

Our approach requires that *each* collaboration is described via connectors and multiple roles in form of a reusable coordination pattern [3]. Coordination patterns specify the real-time coordination behavior of abstract entities by RTSC. These patterns are further used to derive the required component behavior in a process that integrates these design activities with verification.

### 2.1 Real-Time Statecharts

UML state machines are not sufficient to describe complex time-dependent behavior. Therefore, in RTSC time-dependent behavior is modeled using clocks which can be reset when firing transitions or entering resp. leaving a state. Using different clocks enables us to refer to different points in time and therefore permits timing constraints which are not bound to a single state like the *after*-construct.

Similar to Timed Automata, transitions are extended by so called *time guards* and states are associated with *time invariants*. A state must be left before its invariant becomes false.

In contrast to Timed Automata, firing a transition in a RTSC consumes time. Therefore, transitions are associated with *deadlines* specifying points in time when the firing of a transition has to be finished relative to a clock or relative to the point of activation. Besides deadlines, worst-case execution times are included in the model as well. Note, that this is a critical prerequisite to enable an automatic code generation which guarantees the specified timing properties of the model for the generated code.

### 2.2 Patterns and Components

Within our modeling and verification approach for software of complex real-time systems [3], modeling is divided into modeling the interaction between components of the system by reusable *coordination patterns* and modeling the detailed behavior of the components by integrating and refining the behavior of the applied patterns.

A pattern describes communication and therefore consists of multiple communication partners, called *roles*. For communication, roles are linked via connectors. The communication behavior of a role is specified by a RTSC and must satisfy an invariant. The behavior of the connector is described by another RTSC that is used to model channel delay and message losses, which are of crucial importance for real-time systems. Pattern constraints have to be satisfied by the interacting pattern role behaviors. The pattern constraints and the role invariants are annotated to the pattern

respectively its roles using ATCTL<sup>1</sup> formulas.

After the patterns have been specified, concrete software components are built. Components are designed by integrating and refining each role RTSC of the involved patterns. The refinement may not add additional external behavior or block guaranteed behavior. Additionally, it has to respect the guaranteed behavior of the roles in form of their invariants. Additional internal behavior describes the required synchronization of the refined roles. We further refer to the refined roles as (component) ports. The complete system is built by a number of components and patterns which overlap at their ports resp. their roles.

### 3. PROCESS

To complete the presented approach, the outlined modeling capabilities are further extended by model checking and code generation. Model checking proofs that the given constraints hold for the modeled system, whereas code generation ensures that the constraints still hold for the code.

#### 3.1 Integrating Model Checking

In Section 2, we sketched the proposed design approach. When verification is also integrated, the development process consists of the following five steps: (1) design the patterns and their roles, (2) verify each pattern individually, (3) design a component type by choosing patterns in accordance to the required functionality of the component and design the component behavior by refining the role behaviors of these chosen patterns, (4) verify each component type, and (5) build the overall system by instantiating components and connecting their ports in accordance to the corresponding patterns. Note, that steps 1 and 2 have to be repeated for every required pattern. When steps 3 and 4 have already been performed with incomplete sets of patterns, additional orthogonal states that refine the additional roles have to be added incrementally.

Within design step 1, each pattern is associated with a pattern constraint, given as ATCTL formula. In step 2, model checking is used to verify whether the constraint is satisfied by the pattern or not. It is sufficient to check each pattern in isolation since other patterns and components cannot invalidate the constraint due to the employed notion of refinement.

To check the components' correctness, step 4, the role invariants are used. A component is correct, if it fulfills all invariants associated with the refined roles and if it is additionally deadlock free.

Due to the compositional nature of the approach, each system that is composed of verified patterns and components in a syntactically correct manner, step 5, is also semantically correct. An additional sixth step to verify the correctness of the overall system is not necessary [3]. Thus, our approach also allows to verify systems which cannot be verified by non-compositional approaches due to the state explosion.

Another important advantage is that verification steps can be performed on incomplete models. Thus the verification is already performed during the design phase which is realized by a tight integration with the consistency management mechanism of the FUJABA REAL-TIME TOOL SUITE [1] that maintains a continuously up-to-date constraint status. This status is either **true** when the constraint holds, **false** when the

constraint does not hold, or **unsafe** when the related model elements have been changed since the last check.

To maintain consistency between the constraints and structural and behavioral model elements, the following rules are realized by a consistency mechanism: (1) If a pattern or any of its role behaviors in form of a RTSC has been changed, the associated constraints are marked as **unsafe** and all constraints of involved components are marked as **unsafe**, too. The latter marks are necessary because the behavior of each component has to be a refinement of the pattern's roles. (2) If a component or its RTSC has been modified, the related constraints have to be marked as **unsafe**. (3) All constraints marked **unsafe** have to be rechecked. Due to the compositional model checking, only small subsystems have to be rechecked. This usually results in short verification times. Thus, the FUJABA REAL-TIME TOOL SUITE permits to automatically handle these model checking tasks in a background process.

#### 3.2 Code Generation

Verification results are only useful if the implementation of the verified models ensures the obtained results as well. As manual implementation is error-prone, automatic code generation is necessary.

The code synthesis maps each component whose behavior is specified by a RTSC to a set of threads. One of them is the so called periodic *main thread*. This thread periodically checks which transitions are activated, selects one of them, and initiates its execution. As the period of the main thread obviously cannot be zero, activated transition are detected with a delay, which depends on the period, on the applied scheduling approach, and on the applied scheduler.

One main task when implementing a RTSC is the determination of the period which must be as large as possible to achieve a low processor workload, and which must be so small that it leads to delays which are so small that all deadlines are met.

### 4. CONCLUSION

The presented FUJABA REAL-TIME TOOL SUITE enables a smooth integration of design and formal verification during development of networked, real-time systems, and provides a code generation that respects the specified real-time annotations. The employed compositional model checking approach enables an incremental handling of changes or additions during the development.

### 5. REFERENCES

- [1] S. Burmester, H. Giese, J. Niere, M. Tichy, J. Wadsack, R. Wagner, L. Wendehals, and A. Zündorf. Tool Integration at the Meta-Model Level within the FUJABA Tool Suite. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(3):203–218, August.
- [2] S. Burmester, M. Tichy, and H. Giese. Modeling Reconfigurable Mechatronic Systems with Mechatronic UML. In *Proc. of Model Driven Architecture: Foundations and Applications (MDAFA 2004)*, Linköping, Sweden, June 2004.
- [3] 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.

<sup>1</sup>ATCTL is a subset of the timed computation tree logic, which only contains **always** path operators.