Towards the Model-Driven Development of Self-Optimizing Mechatronic Systems*

Holger Giese
Software Engineering Group, University of Paderborn
Warburger Str. 100, D-33098 Paderborn, Germany
hg@uni-paderborn.de

Abstract: Advanced mechatronic systems of the future are expected to behave more intelligently than today by building communities of autonomous agents which exploit local and global networking to enhance their behavior and to realize otherwise not possible functionality. While engineering of mechatronic systems and software engineering for embedded systems, multi-agent systems, and distributed systems are established areas, no solution for the systematic development of the outlined future generation of intelligent, distributed, embedded systems exists today. This is not simply a matter of composing the solutions developed for each of these area as some of their requirements are in conflict: E.g., flexibility and autonomy are to some extent at odds with predictability and safety. We propose to address this challenge by a model-driven development approach which includes several advanced analysis and synthesis techniques. A restricted high level UML model serves as a basis for rigorous validation and verification to address the correctness and safety issues. Analyzing the high level models rather than the code is justified by synthesis techniques, which guarantee that all properties of the high level models also hold for the implementation.

1 Introduction

It is expected that in the next generation of mechatronic systems [BSDB00] we will enhance the functionality and improve the performance by exploiting the ever increasing computing resources and available network technology, e.g., in form of wireless networks.

The information processing of these systems is expected to consist of autonomous agents which coordinate with each other and exploit their context knowledge to enhance their behavior. This will be also true for the control aspects of these systems.

To achieve the required intelligent control behavior, we propose to build self-optimizing technical systems which may endogenously modify their goals in reaction to changes in the environment.\footnote{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.}

Self-optimizing entities are characterized by their ability to (a) sense their environment and state, (b) adjust their goals accordingly, and (c) adapt their behavior
to achieve the chosen goals. Such systems thus result in complex real-time coordination as well as sophisticated quasi-continuous control strategies and their context-dependent reconfiguration. To enable the self-optimization of a technical system, rather sophisticated adaptation schemes are desirable at the control level.

Established approaches for the engineering of mechatronic systems and the software engineering of embedded systems, multi-agent systems, and distributed systems exist. However, for the outlined future generation of intelligent, distributed, embedded systems and their mechatronic aspects no simple combination of these approaches can be sufficient. Conflicting goals such as flexibility and autonomy on the one hand and predictability and safety on the other hand as well as the mere complexity which results from the agent interaction renders the development of the future generation of intelligent, distributed, embedded systems a tough challenge.

To address the outlined challenge with a model-driven development approach, we have identified the following ingredients as essential: (1) Appropriate concepts for the modeling with UML for real-time behavior, advanced agent interaction, and the integration of control theory are required. (2) Advanced tools for the analysis of the models w.r.t. correctness and safety are needed in order to justify the model-driven approach in the domain of embedded, safety-critical systems. (3) The efforts spent on the modeling of complete and detailed models is only reasonable when the transition form the abstract model to the implementation is straightforward. Thus, model-driven development must provide sophisticated synthesis algorithms to derive a consistent implementation at a low cost where possible.

In this position paper, we will discuss each of the identified ingredients and then sketch the proposed solution as planned or already realized within the Fujaba Real-Time Tool Suite.² We start with modeling concepts for real-time and agents in Section 2. In addition, the concepts for the integration of control engineering and software engineering are presented. After reviewing the required analysis techniques in Section 3, the required synthesis support is discussed (see Section 4). Finally, we summarize the position paper and sketch our proposal for the model-driven development of embedded, safety-critical systems.

2 Modeling

2.1 Real-Time Modeling

The ability to specify required real-time behavior in a clear and unambiguous manner also at the model level is crucial for the development of safety-critical mechatronic systems. UML Statecharts permit to specify time dependent triggering of transitions with the after and when construct which usually refer to ticks rather than real-time. In addition, transitions are assumed to have zero-execution time which is also at odds with any real-time processing where deadlines and worst-case-execution-times (wcet) have to be considered.

²http://www.fujaba.de
The UML Profile for Schedulability, Performance, and Time [OMG02] permits to specify several platform-specific real-time processing attributes for threads or processes such as periods or context switch times, but annotations for behavior description techniques of the platform independent model such as Statecharts are not supported. From the large number of object-oriented modeling approaches for real-time systems ROOM [SGW94] has finally found its way into the UML 2.0 specification [Obj03a]. Inspired by ROOM, UML 2.0 thus supports the specification of the structure of complex systems using components with ports and deployment diagrams; however, specific constructs for the modeling of real-time behavior at a higher level of abstraction are still missing.

Therefore the currently available UML CASE tools at most support soft real-time system development. Rhapsody, Rational Rose/RT, Telelogic Tau, or Artisan Real-time Studio Professional only generate code from Statecharts which realize the logical behavior only, while an appropriate mapping onto threads and scheduling parameters to meet required deadlines in form of the synthesis of a platform specific model remains to be determined in a manual process.

To enable the modeling of real-time behavior within platform independent UML models we propose to extend UML Statecharts with clocks, time guards, and time invariants (cf. timed automata [HMP92]) and equip each transition with a deadline which can be specified relative to the firing time or clocks if required. These Real-Time Statecharts [BG03] are supported by the Fujaba Tool and, due to their formal semantics, enable sophisticated analysis of the real-time behavior as well as synthesis which guarantees the real-time deadlines.

On top of this sound foundation, our MECHATRONIC UML approach [BTG04] permits to model complex real-time behavior with component and pattern (cf. [GTB+03]). Besides the components each port/role as well as connectors is equipped with a state machine in form of a Real-Time Statechart to describe the overall real-time behavior for complex systems.

The outlined concepts such as Real-Time Statecharts, patterns, and components are supported by the current version of Fujaba Real-Time.

2.2 Agent Modeling

The multi-agent system paradigm promises to cope with the complexity of the envisioned intelligent mechatronic applications using the agent metaphor. Existing proposals for the modeling of agents with UML are restricted to the abstract information processing level and do not consider real-time or mechatronic aspects.

We propose to achieve the required predictability without ruling out the desired emergent behavior by building the agent modeling concepts on top of the outlined real-time modeling concepts using UML components and patterns. Communities are used as the organizational frame to establish behavioral norms for different agents and Cultures for each community determine the correct interaction in form of pattern [GBK+03, KG04]. The patterns on the one hand permit to ensure the required safe agent behavior using the
later presented concepts to verify UML patterns. On the other hand the degrees of freedom within the pattern roles permit the agents to interact in an intelligent and autonomous manner.

By explicitly grounding all abstract concepts and rules in the concrete entities of an environment model of the mechatronic system, we can further support formal analysis and rapid prototyping. We further propose to separate the requirements and design into largely independent concerns, realized as social structures with behavioral and communicative norms, and carefully composing them for each agent in such a way that the required analytic properties of one aspect is not invalidated by a second aspect (cf. [KG04]).

Story driven modeling [KNNZ00], as supported by Fujaba already, enables to emulate most of the proposed concepts. However, we plan to add direct support for the modelling of cultures and communities to enable the full potential of the approach.

2.3 Integration of Control

The local information-processing units of self-optimizing mechatronic system have to perform a multitude of functions: control code working in quasi-continuous mode controls motions in the plant, error-analysis software monitors the plant in view of occurring malfunctions, adaptation algorithms adapt the control to altered environmental conditions, planning algorithms determine the long-term goals of the agent, different agents have to be safely coordinated, to name but a few.

Each of these functions shows quite different characteristics. While the control strategies in the controller are usually modeled using CAE tools and block diagrams, the coordination between the units is mainly characterized by reactive and proactive real-time behavior, which is best modeled with real-time variants of state machines. The planning, finally, usually requires flexible and powerful structural and behavioral modeling capabilities as offered by modeling approaches such as the UML. A critical prerequisite to the realization of self-optimizing systems is thus an integration between block diagrams as used in mechanical engineering and the UML as employed by software engineers, both at the conceptual and the tool level.

In context of the UML, a RFP for System Engineering [Obj03b] addresses the general integration problem between control and software engineering domain. However, even the Systems Modeling Language (SysML) [Par04], the only relevant proposal, does only provide a very simple integration of differential equations into the UML which does not support reconfiguration. For ROOM an approach for the integration of control engineering block diagrams into some sort of state machine has been proposed in HyROOM [SPP01]. However, the offered integration permits only to reconfigure block diagrams within a superordinate state machine such that reconfiguration is always restricted to a single component/module.

The overwhelming number of additional functions realized by a single agent makes appropriate structuring techniques imperative when designing the corresponding information-processing unit. Therefore, we propose to use the Operator-Controller-Module (OCM)
architecture. The OCM set-up orientates itself by the kind of effect on the technical system: (1) On the lowest level of the OCM, there is the controller featuring an arbitrary number of alternative control strategies. Within the OCM’s innermost loop, the currently active control strategy processes measurements and produces control signals. As it directly affects the plant, it is called motor loop. The software processing is necessarily quasi-continuous, including smooth switching between the alternative control strategies. (2) The controller is complemented by the reflective operator, in which monitoring and controlling routines are executed. The reflective operator operates in a predominantly event-oriented manner. It does not access the actuators of the system directly, but may modify the controller and initiate the switch between control strategies. It furthermore serves as the connecting element to the cognitive level of the OCM. (3) The topmost level of the OCM is occupied by the cognitive operator. On this level, the system can gather information concerning itself and its environment and employ it for the improvement of its own behavior.

To realize this architecture, the controller, which can best be modeled using CAE tools and block diagrams, the reflective operator, which can be modeled with real-time variants of state machines, and the cognitive operator, which requires the flexible powerful structural and behavioral modeling capabilities of the full UML, have to be integrated such that the interaction outlined in the architecture can be modeled. The reflective operator may include selected block diagrams which are used for online diagnosis or reactions to certain events. Additionally, the cognitive operator may evaluate a block diagram model in an asynchronously running thread for simulation and prediction purposes.

To support the modular reconfiguration of the internal structures of the controllers, we developed hybrid UML components and a related hybrid Statechart extension for the UML [BGO04]. The hybrid components support the design of self-optimizing mechatronic systems by allowing specification of the necessary flexible reconfiguration of the system as well as of its hybrid subsystems in a modular manner.

An XML Encoding of the hybrid components is currently under development in a student project and a Bachelor thesis to integrate the CASE tool Fujaba Real-Time Tool Suite and the CAE tool CAMeL3. Hybrid Statecharts are additionally realized within the student project.

3 Model Analysis

3.1 Safety

A unwanted consequence which results for the outlined trend towards complex interconnected technical systems are serious problems to ensure the system safety. Due to the complexity as well as the unpredictable nature of self-optimizing mechatronic systems, applying standard approaches is by no means sufficient any more.

3www.ixtronics.de
The current and forthcoming UML versions do not directly support safety-critical system development. Available hazard analysis techniques on the other hand have their origin in the hardware world and do not provide the required degree of integration with software design notations. They assume a very simple hardware-oriented notion of components and therefore do not directly support the identification of common mode faults. Some more advanced approaches [PMRSH01, KLM03, Gru03] support a compositional treatment of failures and their propagation, but still a proper integration with concepts like deployment and the more complex software interface structure is missing.

In [GTS04] our approach for the compositional hazard analysis of the outlined UML models with components and patterns which narrows the described gap between safety-critical system development and available UML techniques is outlined. It builds on the foundation of failure propagation analysis [FMNP94] and permits automatic quantitative analysis at an early design stage. The failures can be modeled as detailed as required using a hierarchical failure classification where correct refinement steps ensure the complete coverage of all possible failures. The approach permits to systematically identify which hazards and failures are most serious, which components or patterns require a more detailed safety analysis, and which restrictions to the failure propagation are assumed. We can thus systematically derive all safety requirements, which correspond to required restrictions of the failure propagation of a single component, pattern, or a system of components and pattern in the UML design.

The presented concepts are currently implemented in Fujaba within a Bachelor and a Master thesis.

3.2 Correctness

The outlined advanced multi-agent systems have, in contrast to classical control systems, rather complex run-time behavior. Therefore, standard means for verification and validation such as testing are by no means sufficient to ensure that the system correctly fulfills the safety requirements identified during hazard analysis.

Knapp et al. present in [KMR02] a tool called HUGO/RT. Within this tool, models are described by UML state machines. The properties to be checked are given as scenarios written as sequence diagrams extended with time annotations. For verification, HUGO/RT transforms the Statecharts into Timed Automata and the sequence diagrams into Observer Timed Automata and applies the model checker Uppaal. The approach of Diethers and Huhn [DH04] is similar to this, but supports the commercial CASE tool (Poseidon).

Another project that aims at modeling and verifying real-time and embedded systems with UML is the OMEGA IST project. The project does not support the complete UML language. Instead, a subset of the UML which is essential for the modeling of industrial real-time applications [DJVP03] is defined. In addition, a subset of the UML is extended by some timing constructs [GOO03] which are necessary when modeling real-time systems. The integrated validation tools support simulation, verification of the properties and automatic test generation.
The applicability of model checking is however rather limited when it comes to the verification of complex distributed embedded real-time systems due to the state space explosion problem. Only the OMEGA IST project tries to tackle this problem by techniques based on data flow analysis, slicing methods, and simple forms of abstraction, but no compositional model checking approach is provided.

Due to their complexity and history dependent behavior, the addressed complex self-optimizing systems cannot be model checked directly. We therefore propose to ensure their correctness using the composition of the following individual steps: (I) Model checking of Real-Time Statecharts including their real-time behavior. (II) Compositional model checking for the distributed coordination of multiple reflective operators. (III) Concepts for the safe integration of the cognitive operator (IV) and finally rules for syntactically checking the correct embedding of hybrid components (controller) into the reflective operator.

(I) As described in [BGHS04], to model check Real-Time Statecharts, we at first map them to HUppaal and then use the tool Vanilla to transform them to timed automata as required by Uppaal [DMY02].

(II) Our approach addresses the state explosion problem for a set of interconnected reflective operators, using compositional model checking and an integrated sequence of design steps (cf. [GTB\textsuperscript{+}03]). These steps prescribe how to compose complex software systems from domain-specific patterns which model a particular part of the system behavior in a well-defined context. The correctness of these patterns can be verified individually because they have only simple communication behavior and have only a fixed number of participating roles. The composition of these patterns to describe the complete component behavior and the overall system behavior is prescribed by a rigorous syntactic definition which guarantees that the verification of component and system behavior can exploit the results of the verification of individual patterns. Compositional model checking and role refinement thus enable the verification of the real-time coordination of large, complex mechatronic systems which result from the interplay of the reflective operators.

(III) To also take the full behavior of the cognitive operators with all their complexity and history dependent evolution into account is not feasible when a complete automatic formal verification is intended. We therefore propose to exploit the architectural separation between the cognitive operator and the reflective operator instead to ensure a safe behavior (cf. [GBK\textsuperscript{+}03]). Thus, the reflective operator filters the input of the cognitive operator to prevent that its unpredictable nature can result in an unsafe operational behavior.

(IV) Self-optimization results in rather complex reconfiguration schemes within the reflective operators and controllers that are composed in a modular manner. We developed an approach [GBSO04] for the modular hierarchical composition of event-based and quasi-continuous behavior where simple consistency checks which are applied for each embedding hybrid components are sufficient to ensure that the reconfiguration only results in correct configurations and that the verified event-based real-time behavior still holds.

The model checking of Real-Time Statecharts, the compositional model checking, and the abstraction from the effects of the cognitive operator are available in the Fujaba Real-Time Tool Suite. The outlined consistency check for the hybrid embedding is planed to be realized within a Master thesis.
4 Advanced Synthesis

During the model-driven development an abstract platform independent model (PIM) is first developed and then refined towards a platform specific one. All properties guaranteed by the PIM have to be preserved by the refinement towards a platform specific model (PSM) and the final code. Thus tool support for the transition from a PIM to a specific PSM which synthesizes required attributes where possible as well as code synthesis for the final target platform is required. Otherwise, ensuring that the high level properties present in the UML models are still present in the implementation becomes in most cases impossible or at least a very tedious task.

The UML Profile for Schedulability, Performance, and Time [OMG02] defines general resource and time models which are used to describe the real-time specific attributes of the modeling elements such as schedulability parameters or quality of service (QoS) characteristics. In terms of MDA, besides a PIM, a more concrete PSM can be specified by using the extensions of the profile. This PSM can be later used for the required model analysis and code generation. However, it remains an open question in the UML profile how all required details of the PSM are determined. In a scenario where the developer maps his model onto the technical concepts such as threads and periods manually, we still have the problem that this mapping results in an iterative manual process of testing and adjusting the model until the real-time constraints are met. Consequently, current CASE tools do not provide sophisticated synthesis of PSM from PIM or code generation which considers resource constraints and guarantees that the real-time constraints are met.

While a number of approaches for synthesis and scheduling analysis for the PSM level exists (e.g., [FGHL04, HSG01, GKWS03]), there are only a few synthesis approaches which support the developer when refining a PIM towards a PSM. Modecharts are a suitable high-level form of state transition systems for the specification of real-time systems, but available code generation does not consider the deadlines or periods [PMS95]. In [ADF01], scheduling analysis and code generation for timed automata with tasks with WCETs and deadlines associated to locations are presented. However, the presented approach does not take the transition delays into account, arguing that these delays are small compared for the WCETs.

To close the above identified gap between the platform independent model at a high level of abstraction and the platform specific model and the implementation which fulfills the required real-time constraints, we have developed a synthesis algorithm [BGS03]. For a restricted subset of UML and Real-Time Statecharts, the algorithm automatically partitions the model to a PSM and code generation, which take CPU time sharing on a single microprocessor into account.

As the PSM and code are synthesized automatically the (platform dependent) WCETs of this implementation are well-known. The automatic partitioning in the PSM respects the WCETs of the local side-effects as well as the WCETs for the implementation of the statechart behavior and the specified deadlines. Therefore the algorithm can guarantee that all real-time requirements are met, which makes an additional analysis unnecessary and avoids a costly iterative manual process. In addition, an integration of quasi-continuous
and event-based discrete models is required to enable reconfiguration. We thus have developed a shared execution framework which supports efficient reconfiguration and modular code generation [BGGO04].

A first version of the outlined high-level code synthesis which includes the partitioning and mapping onto threads has been realized for Java Real-Time. An extension which permits to describe mapping decisions using an explicit platform dependent model and support for C++ is currently under development. The integration of the code execution scheme for UML models and quasi-continuous blocks is currently realized for the non-distributed case in a student project.

5 Summary & Conclusion

The proposed approach addresses the challenge of model-driven development of safety-critical embedded systems as outlined in the introduction by integrating several advanced analysis and synthesis techniques as well as control theory related quasi-continuous approaches with the model-driven development with UML. A high level UML model with some minor extensions serves as a basis for rigorous validation and verification to ensure that required correctness and safety are guaranteed. Synthesis in form of automatic code generation, which guarantees that the all verified properties of the high level model also hold for the implementation justifies that the analysis is done on the more abstract models.

For the outlined model-driven development of self-optimizing mechatronic systems we propose the following process consisting of three main parts (cf. [BTG04]):

In the first part, the safety related requirements are systematically derived using hazard analysis and failure propagation models. Then, individual coordination patterns are developed which realize the non local safety requirements. If it has been successfully verified that the pattern ensure the required safety properties, it is then added to a pattern library.

In the second part, the mechatronic agents are built using the verified coordination patterns stored in the library of patterns by refining and coordinating the pattern roles such that the verified real-time properties are preserved. In the next step further components (e.g. hybrid ones) are embedded into the superordinated component. Simple consistency checks ensure again that the verified real-time properties of the coordination patterns are still valid in spite of the embedding.

As the last part, the PSM and the source code are synthesized for the structure and behavior of the UML model.

The outlined model-driven development is further supported by a sophisticated consistency management subsystem for the integration of different models [BGN+04]. For example, the compositional model checking approach has been realized within Fujaba offering a tight integration for managing the required compositional verification steps using the consistency management subsystem (cf. [BGHS04]) such that an incremental and iterative design and verification process becomes possible.
The presented approach provides the most essential ingredients which are required to engineer the envisioned complex self-optimizing mechatronic systems. Our believe is that the presented solutions address the most crucial problems which have to be resolved to enable the model-driven development of the advanced embedded systems of the future.

Acknowledgements

Thanks to Florian Klein for his comments on an earlier version of this position paper.

References


