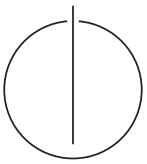
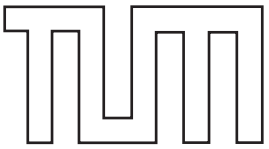


TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Software & Systems Engineering
Prof. Dr. Dr. h.c. Manfred Broy



SPES 2020 Deliverable D1.1.A-2

Towards an Integrated Modeling Theory

A First Step in the Integration of the Semantic Foundations of FOCUS, Rich Components, and Mechatronic UML

Author: Alexander Harhurin (TUM-SSE)
Daniel Ratiu (TUM-SSE)
Judith Thyssen (TUM-SSE)
Jörg Holtmann (UPB)
Hary Hungar (OFFIS)
Eike Thaden (OFFIS)

Version: 1.1
Date: January 11, 2010
Status: DRAFT

About the Document

The aim of this paper is to compare three modeling approaches that are relevant within the SPES context, namely the FOCUS approach from TU-München, the Rich Components approach of OFFIS, and the Meachatronic UML of Uni Paderborn. Hereby, we focus on the comparison of the underlying modeling theories, as an essential prerequisite for the integration of the modeling approaches.

In this paper, we motivate the use of an integrated modeling theory, shortly introduce the three modeling approaches, compare them with respect to their underlying modeling theories, and sketch the next steps towards their integration into a comprehensive modeling theory that can be used as semantic basis for the whole SPES project.

Contents

1 Introduction

Even if adopted in practical development of embedded systems today, model-based development approaches often fall short due to the lack of powerful enough modeling theories and missing integration of theories, methods and tools. By *modeling theory* we mean the semantic foundation of a modeling language.

Many critical issues arise due to a missing, conflicting or inappropriate semantic foundation of the modeling languages used today:

- Many times, the semantics of a modeling language is only specified as prose (if at all) which leads to ambiguities in the interpretation of the created models and to inconsistencies between different implementations of the language in different tools.
- For other languages there exist several different semantic foundations. This results in different dialects of the same modeling technique which prevents an uniform treatment of the built models as well as the translation of models written in one formalism to another formalism.
- Moreover, different models applied in different phases of the development process are based on separated and unrelated modeling theories, which makes the transition from one model to another unclear and error-prone.

Within the SPES project, there exists different modeling techniques based on different modeling theories. As described before, this fragmentation and isolation of the semantic foundations prevent the integration of models written in the different modeling languages, the verification of their consistency, and the (automatic) translation of one model to another. Thus, the integration of the different semantic foundations is an indispensable prerequisite for a deep integration of the different modeling worlds and, consequently, for an integrated model-based development approach as overall result of the whole SPES project.

The integration of the semantic foundations aims at an *unified modeling theory* giving a consistent semantic foundation for all systems modeling approaches developed within SPES-project. Thereby, the modeling theory must be strong and expressive enough to support the whole development process starting from initial requirements down to a running system and to model all relevant aspects of hardware and software architectures of a system such as structuring software deployment, description of tasks and threads, as well as modeling behavior aspects of hardware.

In [HHR09] we introduced the FOCUS approach for the development of reactive systems. The aim of the current paper is to shortly compare the FOCUS approach to other approaches that are relevant within the SPES context, namely the Heterogenous Rich Components (HRC) approach of OFFIS, and the Meachatronic UML of University of Paderborn. Hereby, we focus on the comparison of the underlying modeling theories, since their integration is the essential basis for an integrated modeling approach desired in SPES.

It is important to remark that the modeling approaches can not be compared without respect to the purpose that they were designed for. Depending on the purpose an approach has to serve, a different mathematical foundation can be adequate. For example, if the models serve only as documentation and communication means, a strong mathematical foundation is less

important. However, to use the full power of model-based development including verification and code generation, a thorough mathematical foundation is essential. Moreover, the decision for a certain modeling theory also depends on the system to be developed. If the focus lies on the software that should run on discrete controllers on a discrete timing model is sufficient. However, if the theory aims at modeling the environmental/physical relations then a continuous timing model is needed.

In this paper, we sketch the differences between the different modeling approaches and show how they could be integrated. Therefore, we shortly recapitulate the different modeling approaches in Sections 2, 3, and 4. Thereafter, in Section 5, we compare the approaches with respect to their underlying modeling theory and sketch the next steps towards their integration into an unified modeling theory for the whole SPES project.

2 Focus

The FOCUS modeling approach introduced in [BS01, Bro05] aims at modeling two fundamental, complementary views onto multi-functional systems:

- User Functionality Hierarchy: A structured view onto the overall functionality offered by a multi-functional system by decomposing the system functionality into a hierarchy of interrelated user functions. We speak of the *functional specification*. The *service hierarchy* aims at capturing all software-based functionality (*services*) offered by a system to its users. This hierarchy specifies the system behavior from a black-box perspective by capturing the family of services offered by the system. It aims at understanding how the services are structured and how they depend on and interfere with each other [BKPS07].
- Logical Component Architecture: Decomposition of the system into a network of components that mutually cooperate to generate the behavior specified by the user functionality hierarchy. We speak of the *design*. In a *component network*, the system functionality is specified as a distributed system of interacting components. Via their interaction, the components realize the observable behavior described in the service hierarchy.

In the following we put this modeling theory in a larger context according to its underlying timing model and its communication/synchronization mechanisms between different components.

Communication Paradigm. Regarding communication – i. e., the way information can be exchanged between components – one can distinguish between *implicit communication* (e. g., via shared variables), *explicit event based synchronous communication* and *explicit message based asynchronous communication*.

FOCUS is based on explicit message based asynchronous communication. In this class there is an explicit communication mechanism with clear distinction between sender and receiver. The communicating partners are decoupled, i. e., while the receiver has to wait until a message is available to be read, the sender can write a message without delay.

Timing Model. Considering possibilities at what point in time "relevant" activities are allowed to happen during the system run, the following classification can be done: In *time continuous* systems, relevant activities can evolve continuously. In *time discrete* systems, the time line is divided into not necessarily but possibly equal intervals and the relevant activities only occur at points of time of the time grid. In *event discrete* systems, events can occur at any point of a real time line.

FOCUS assumes a simple model of discrete time, where time advances in ticks. Therefore, events within a single tick can not be differentiated. However, extending the theory to more complex models of real or continuous time can be easily achieved following [Bro97].

Synchronicity of Time. Different theories differentiate whether there is a global clock which defines a global time for all parts of the system, or if the different components have their local clock. In the second case further synchronization mechanisms are necessary. FOCUS assumes a global clock.

For more details about the issues presented above please refer to the SPES deliverable D1.1.A [HHR09].

3 Heterogeneous Rich Components

In addition to traditional static interfaces of UML-components that only define the interaction points of components, richer information is exposed on the boundaries of (Heterogeneous) Rich Components (HRCs) [DVM+05], in terms of contracts. Attached to a component, contracts abstract dynamics constraints on the component in terms of assumption-promise pairs, with the meaning that a promise offered by the component is guaranteed only if the corresponding assumption is fulfilled in the system context. The concept of HRC allows to combine components modeled in different front-end tools (Rhapsody, Matlab Simulink, etc.) into one abstract model with one well-defined semantics and to verify their behavioral requirements against each other.

According to [JMM08] RCs comprise the features presented below.

Different layers of abstraction. Different layers are identified in correspondence to different architectural abstractions of an embedded system. Examples of such architectural layers are: the functional layer that abstracts the functionality of the system; the ECU layer that abstracts the system as a network of tasks and signals between them on the one hand and a network of ECUs and buses on the other hand. RCs serve as the basic syntactical units of construction for all layers.

Structure of Systems. As the basic unit of construction, an RC represents the abstraction of some functionality and its properties. More specifically, an RC consists of the following ingredients:

- a (static) interface that defines the interaction/access points to the environment

- a set of contracts that specify the dynamic functional and non-functional properties of the Rich Component
- an implementation (e.g. C code, VHDL, etc.) that realizes the functionality (optional)

Interface. The interface of a RC is a set of interaction points aggregated into ports. Both service-oriented and data-oriented interaction points are supported. An RC may either provide or require a service. In both cases, the signature of the service, namely its name, parameters and return type, are depicted in the interface. In addition to services, flows are also defined as interaction points for data exchange. A *discrete* flow defines a data exchange point where there is always a value available and the values evolve discretely. A *continuous* flow defines a data exchange point where there is always a value available and the values evolve continuously. An *event* flow defines a data exchange point where only at certain time instants there is a value available and the values may change from instant to instant.

Behavior. One can distinguish between behavior requirements and actual behavior models. Contracts are used to specify behavior requirements for Rich Components. They can belong to different viewpoints (e.g. real-time, safety, etc.) to allow separation of concerns. Dominance (implication) is the central relation between contracts.

It is possible but not necessary to specify an implementation for a Rich Component. This implementation should satisfy the contracts attached to the RC. Implementations can be given in the form of code (generated or hand-written C code, etc.). Semantically, an implementation is a set of runs which may be defined by specific extended automata.

Contracts are built using state machines, which are hybrid automaton with a well defined C-like action language to express actions on the transitions and inside locations. However, according to the agreement of a series of meetings between TUM, OFFIS and Uni Padeborn, timed rather than hybrid automata are used in the praxis.

Timing Model and Synchronicity of Time. The underlying semantics of contracts uses dense time and distinguishes discrete steps and continuous evolutions. Time is evolving during continuous evolutions while discrete changes take no time. Currently a unique global physical time is used. Clocks are modeled as ordinary continuous variables with trivial evolution (e.g. for a clock c the differential equation $\dot{z} = 1$ is always active).

Communication Paradigm. Communication in HRC is synchronous. It happens continuously during continuous evolutions, and (also) instantly at discrete events. The latter means that a transition where an event is sent, all transitions which are waiting for that event are taken at the same time.

Probabilism Though the full semantical spectrum of HRCs includes probabilism (and its relation to nondeterminism), this is not expected to become important for SPES.

4 Mechatronic UML

Mechatronic UML is an UML-based approach for modeling mechatronic systems that combine technologies from mechanical engineering, electrical engineering, and computer science. For a broad acceptance (partly extended) UML diagrams and constructs are used, but enriched with underlying formal semantics. Mechatronic systems are real-time systems and contain both discrete control modes as well as implementations of continuous feedback controllers. Moreover, they are distributed. Different parts require coordination in order to act as a whole. Thereby, each part must be reconfigurable in order to comply with different roles that result from the coordination [BGT05]. Below are described the most important features of Mechatronic UML.

Structuring the System. To avoid the state explosion problem on verification of the whole system and to allow a compositional verification of the single parts of the system, the system is divided into structure and communication.

Coordination/Communication Patterns All communication relationships used in the system are modeled by *Real-Time Coordination Patterns*. A pattern consists of roles as well as a connector between these roles. Each role is described by a protocol automaton, more precisely an extension of statecharts which allows the specification of reactive real-time behavior (see below). For the connector it is possible to specify QoS-characteristics of the underlying transportation protocol (e.g. channel delays or loss of messages) with such an extended statechart. The overall communication behavior is described by the role behaviors, which exchange messages via the connector. The role behaviors contain parts of nondeterministic behavior, which is resolved when the pattern is applied on a component [GTB+03].

Component Model The Mechatronic UML uses UML component diagrams to specify the structure of the system. It distinguishes between the components and their instances at runtime by providing a type concept. The internal data structure of a component is specified with a class diagram. Components possess ports describing the interaction with the components' environment with an automaton. The ports have interfaces describing the messages, which can be exchanged over the ports/automata. Each port of a component is assigned a role of a coordination pattern from the pattern catalog. The port has to refine the pattern role by resolving the nondeterminism of the role automaton. The internal behavior of the component is described by an automaton synchronizing all port automata. Components can contain further components, so a system consisting of these components is spanning a component hierarchy. Moreover, a component can be discrete or continuous. Discrete components are the ones described above, while continuous components are abstractions of continuous feedback-controllers whose behavior might be described with a differential equation, for example. These continuous components are only embedded at the leaf level in the component hierarchy. A discrete component embedding both discrete and continuous components are also called hybrid components, as we see below. After all needed components have been fully specified, different instances of each can be created to employ them in a system at runtime [BGT05, GTB+03].

Specifying the Discrete Behavior. *Real-Time Statecharts (RTSCs)* specify the behavior of the discrete parts of the system [GB03], that is, the communication behavior of the patterns as well as the components' internal behavior. RTSCs combine constructs from timed automata like clocks, time guards, time invariants, worst time execution times, and deadlines with constructs from statecharts like hierarchical states or parallelism. Transitions make use of events, guards, time guards, resets, priorities, and synchronization. States make use of entry/exit operations at the entrance and the exit of a state, respectively, and of do operations that are periodically invoked during the residence in a state [GB03, p.24]. To continue the model-based approach, these operations are specified with a graph transformation formalism, which operates on data structures described by class diagrams and enables code generation. Since it is not trivial to determine WCETs for dynamic, object-oriented data structures, a method was developed to calculate and optimize WCETs directly on the graph transformations. Together with these model-based operations, C++ or Real-Time Java code for the whole RTSC can be synthesized. The semantics of RTSCs are specified by a mapping to Timed Automata, which also allows verification of the timed behavior by model checking.

Specifying the Hybrid Behavior. The hybrid behavior is modeled with *Hybrid Reconfiguration Charts (HRCs)*, an extension of RTSCs where each discrete state is associated with a configuration of discrete components or continuous blocks, leading to hybrid components [GBSO04]. A HRC can be applied as synchronization chart within each hybrid component. The formal semantics of HRCs are described by a mapping to Hybrid I/O Automata. To avoid a state explosion problem of the verification, these Hybrid I/O Automata are not model checked, but different verification techniques are applied. The correct embedding of component instances in a containing component instance is checked for each state syntactically by looking up the allowed configurations of the containing component type. As HRCs are extensions of RTSCs, the timing behavior is verified by model checking. After all, the continuous behavior is analyzed in the control engineering domain.

Compositional Verification. The division of the system into Real-Time Coordination Patterns and component types allows the verification by model checking the single parts of the system. First, a pattern is verified to show a correct communication behavior between the future components. Then, components with ports and interfaces are build, and the pattern roles are assigned to the ports. The refinement relation between roles and ports ensures that all patterns are employed correctly to the components. Finally, the components' internal behaviors, synchronizing all the refined roles, are verified. A system is build up only with instances of these verified components, leading to a correct system wrt. safety properties without running into scaling problems [GTB+03].

Timing Model. As mechatronic systems are distributed systems and thus contain asynchronous, event-based processes, in [BGHS04] a continuous timing model for the patterns was introduced to use its advantages like having an infinitely small interval between two events. A discretization can occur on several occasions and with different clock cycles for each target, for example, if a simulation for a system (part) is needed or a possible target hardware is known. In these cases, the verification properties must be preserved.

Synchronicity of Time. Mechatronic UML also has the common synchronization assumption induced by Timed Automata, that is, there are local clocks without clock drift. But the fact, that there is a local clock for each automaton, enables in turn to model and thus verify the clock drift explicitly [GHH06].

5 Towards an Integrated Modeling Theory

In Table 1, we compare the semantics of the different modeling approaches with respect to the following criteria: basic formalism used to express the behaviour, timing model, and communication paradigm, basic building blocks, compositionality, and refinement.

Approach	Basic Formalism	Timing Model	Communication Paradigm	Basic Building Blocks	Compositionality	Notion of Refinement
FOCUS	Stream-processing functions	time discrete	asynchronous communication	components	yes	property refinement, interaction refinement
Rich Components	Timed automata (hybrid automata)	event-discrete/ continuous time and continuous evolutions	synchronous communication	components	yes	dominance, satisfaction
Mechatronic UML	Timed automata (hybrid automata)	event-discrete/ continuous time	asynchronous communication (synchronous within one RTSC)	components and communication patterns	yes	component/ interface refinement(???), role/behavior refinement

Table 1: Comparison of the modeling approaches

Scope of the SPES unified modeling theory. Even if we are aware of the fact, that for all of the presented approaches a lot research activities are currently being performed spanning over a brought variety of different aspects including hybrid systems, probabilistic systems or dynamic reconfiguration, after a series of meetings between TUM, OFFIS and Uni Padeborn, we agreed to neglect these complex aspects and to concentrate on discrete systems as basis for the first proposal of a unified modeling theory for the SPES project.

First stage of the integration. As a consequence, the next step towards an integrated modeling theory must be to find a mapping between the timed automata underlying the Rich Component and Mechatronic UML approach and the stream-based FOCUS approach. In [HHR09], we have already shown, how the stream-processing FOCUS functions can be translated into I/O-automata. Next, we have to show how timed automata and I/O-automata can be integrated.

Next stages of the integration. As mentioned before, we agreed not to consider continuous or hybrid systems in the first step. However, we are aware of the relevance of modeling continuous and hybrid aspects in the domain of embedded systems. Thus, we are planning to elaborate appropriate concepts and to extend the modeling theory accordingly in a second step,

if necessary. In [Bro97], for example, it is already sketched that and how the FOCUS theory can be extended to more complex models of real or continuous time. Also, the integration of different, not necessarily orthogonal viewpoint specifications needs to be considered.

References

[BGHS04] Sven Burmester, Holger Giese, Martin Hirsch, and Daniela Schilling. Incremental design and formal verification with UML/RT in the FUJABA real-time tool suite. In *Proc. of the International Workshop on Specification and Validation of UML Models for Real Time and Embedded Systems, SVERTS2004, Satellite Event of the 7th International Conference on the Unified Modeling Language, UML2004*, pages 1–20, October 2004.

[BGT05] Sven Burmester, Holger Giese, and Matthias Tichy. Model-driven development of reconfigurable mechatronic systems with mechatronic uml. In Mehmet Aksit Uwe Assmann, Arend Rensink, editor, *Model Driven Architecture: Foundations and Applications*, volume 3599 of *Lecture Notes in Computer Science (LNCS)*. Springer Verlag, 2005.

[BKPS07] M. Broy, I.H. Kruger, A. Pretschner, and C. Salzmann. Engineering automotive software. *Proceedings of the IEEE*, 95(2):356–373, Feb. 2007.

[Bro97] M. Broy. Refinement of time. In M. Bertran and Th. Rus, editors, *Transformation-Based Reactive System Development, ARTS'97*, number LNCS 1231, pages 44 – 63. TCS, 1997.

[Bro05] Manfred Broy. *Service-Oriented Systems Engineering: Specification and Design of Services and Layered Architectures - The Janus Approach*, volume 195, pages 47–81. Springer Verlag, July 2005.

[BS01] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001.

[DVM+05] Werner Damm, Angelika Votintseva, Alexander Metzner, Bernhard Josko, Thomas Peikenkamp, and Eckard Böde. Boosting re-use of embedded automotive applications through rich components. In *Proceedings, FIT 2005 - Foundations of Interface Technologies*, aug 2005.

[GB03] Holger Giese and Sven Burmester. Real-time statechart semantics. Technical report, University of Padeborn, 2003.

[GBSO04] Holger Giese, Sven Burmester, Wilhelm Schäfer, and Oliver Oberschelp. Modular design and verification of component-based mechatronic systems with online-reconfiguration. *SIGSOFT Softw. Eng. Notes*, 29(6):179–188, 2004.

[GHH06] Holger Giese, Stefan Henkler, and Martin Hirsch. Analysis and modeling of real-time with mechatronic uml taking clock drift into account. In *Proc. of the International Workshop on Modeling and Analysis of Real-Time and Embedded Systems (MARTES), Satellite Event of the 9th International Conference on Model Driven*

Engineering Languages and Systems, MoDELS/UML2006, Genova, Italy, volume 343 of *Research Report*, pages 41–60, University of Oslo, October 2006.

[GTB+03] Holger Giese, Matthias Tichy, Sven Burmester, Wilhelm Schäfer, and Stephan Flake. Towards the compositional verification of real-time uml designs. In *Proc. of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-11)*, pages 38–47. ACM Press, September 2003.

[HHR09] Alexander Harhurin, Judith Hartmann, and Daniel Ratiu. Motivation and Formal Foundations of a Comprehensive Modeling Theory for Embedded Systems. Technical Report TUM-I0924, Technische Universität München, 2009.

[JMM08] Bernhard Josko, Qin Ma, and Alexander Metzner. Designing embedded systems using heterogeneous rich components, 2008.