ISBN 978-82-326-4922-8 (printed ver.)
ISBN 978-82-326-4923-5 (electronic ver.)
ISSN 1503-8181

Norwegian University of Science and Technology

Doctoral thesis

Fenglin Han

Doctoral theses at NTNU, 2020:287

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Doctoral theses at NTNU, 2020:287

Fenglin Han

MODEL-DRIVEN ENGINEERING OF COMPLEX SYSTEMS

Supporting the Design of Reactive Systems with Augmented Modeling and Verification Mechanisms

NTNU
Norwegian University of Science and Technology

Fenglin Han

MODEL-DRIVEN ENGINEERING OF COMPLEX SYSTEMS

Supporting the Design of Reactive Systems with Augmented Modeling and Verification Mechanisms

Thesis for the Degree of Philosophiae Doctor

Trondheim, October 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

© Fenglin Han

ISBN 978-82-326-4922-8 (printed ver.)
ISBN 978-82-326-4923-5 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2020:287

Printed by NTNU Grafisk senter

Abstract

This thesis motivates the need for the Reactive Blocks approach for model-driven Complex Systems Engineering (CSE). With the booming of the Internet economy and development of the Internet of Things (IoT), interconnected applications have penetrated into areas including manufacturing, management, and all aspects of human life. Embedded devices and internet-based services are increasingly inter-connected, and as the applications and services grow, the devices are intertwined into a complex system. This approach comprises a core process that constitutes the practice of model-driven development, component-based distributed system synthesis and derivation, software verification, modeling and verification augmentation for a variety of properties, such as real-time, probabilistic properties of complex systems.

We constrain CSE to distributed and concurrent reactive software systems that are prevalent in many application domains, such as avionics, consumer electronics, robotics and new emerging sensor networks. These domains need to address a variety of properties in software and hardware. The efforts involved in implementing a system correctly, consume the majority of the development time and cost of complex systems; thus, many design and verification methods have been devised and applied to software development to ensure correct implementation. The Reactive Blocks approach, initiated by the network systems group in the Department of Information Security and Communication Technology, Norwegian University of Science and Technology, is designed to use reusable building blocks to build systems with significantly reduced development time and cost while at the same time guaranteeing the functional accuracy and transparency. Reactive Blocks takes the UML activity and state machine diagrams as the main design unit and provides a visual programming interface, while the implementation details are hidden in the diagrams. Reactive Blocks allows more than specifying just functional properties, but also QoS properties like real-time and performance. To empower Reactive Blocks with QoS properties analysis in our approach, we imported and analyzed a graph representation of our approach. Based on the graph representation, we extended the verification of Reactive Blocks with augmented extensions and applied rich formal method based analysis.

In the first part of the thesis, we analyze the graph representation in Reactive Blocks. To analyze the graph representation, we imported graph transformation rules for automatically analyzing the decomposition and automatic remedy of design errors in the system specification. This part involves two papers with one addressing the choreography model transformation via graph transformation rules and the other automatic system design error detection and remedy.

Second, the modeling capabilities of Reactive Blocks are augmented and extended to include new formalisms. New system requirements are addressed using extended state transition diagrams. To describe the real-time constraints in building blocks, we extend and augment the external state machine (ESM) to a real-time ESM (RTESM) with clock variables and guard notations. Further, the augmentation capabilities consider possibilities and formalize the probabilistic RTESM (PRTESM). This part involves three papers addressing real-time modeling and verification, a performance index for building blocks, and probabilistic modeling and verification of reactive systems respectively.

Third, with its extended modeling capabilities, the Reactive Blocks approach can be used in the modeling and verification of probabilistic real-time systems which are typical in cyber-physical systems, such as robots and aviation control systems. This is discussed in two papers. In one of them, we summarize a tool chain for the formal development of controllers for hybrid systems that need to fulfill both real-time and spatial behavior properties. In the other one, we depict a hybrid control system background and combine the simulation tool JEmula with our reactive system design and analysis. We also use the BeSpaceD tool for the 802.11 WLAN wireless signal strength and coverage area analysis, which facilitates the safe control of robots.

Our CSE approach has a strong focus on formal methods. We use timed automata and probabilistic timed automata for interpreting the extended specification. In particular, we apply Timed Computation Tree Logic (TCTL) and Probabilistic Timed Computation Tree Logic (PTCTL) to verify timed probabilistic properties of the specification, and such formalisms (RTESM, PRTESM) inherit the compositional and incremental verification characteristics of our Reactive Blocks paradigm. We integrated several tools to support our approach, including AGG and HENSHIN for graph based model transformation, and we used the popular real-time verification tools UPPAAL and PRISM for time and probabilistic properties verification. These tools were integrated using self-developed software facilities that automatically migrate the syntax of different formalisms. The spatial properties simulation and verification were achieved by integrating Reactive Blocks with the spatial property verification tool BeSpaceD. We built the control function building blocks of robot controllers and simulation environments that generate output for the BeSpaceD tool. The embedded robot control system addresses the real-time and probabilistic properties of hybrid system. Traces of robot movements were analyzed, and certain spatial safety properties were verified by the BeSpaceD tool.

Keywords: Reactive Blocks, concurrent distributed system, software verification, model-driven software development

Preface

This thesis is submitted in fulfilment of the requirements for the degree of the philosophiæ doctor (Ph.D) at Norges Teknisk-Naturvitenskapelige Universitet (NTNU). The work for the thesis was done at the former Department of Telematics of NTNU, which was merged into the Department of Information Security and Communication Technology, and under the supervision of *Professor Peter Herrmann* and co-supervised by *Professor Rolv Bræk*.

The thesis work was performed within the context of the Research Council of Norway *(RCN)* sponsored project "Infrastructure for Integrated Service" *(ISIS)*, which was a multidisciplinary effort of the Telenor Research, TellU, the Department of Information Security and Communication Technology of NTNU and the University of Agder. The *ISIS* project aimed at providing a real model-driven process from requirements capturing via design synthesis and analysis to code generation and service execution. The approach is supported with tools and service execution platforms that substantially improve industrial service engineering on a seamless infrastructure.

While in the writing process of this thesis, the science-fiction novel 'The Three-Body Problem' has won the Hugo award in 2015. Here, I like to congratulate the Chinese writer Liu Cixin for his achievement. 'The Three-Body Problem' is the only novel I read during my PhD study and it has been long time since I last read science fiction novels. I like to thank Liu Cixin for the joyness he brought to me during the thesis work in Trondheim and piles of papers in front of my desktop. In an interview, Liu said: "The future in the people's eyes is full of attractions, temptations and hope. But at the same time, it is also full of threats and challenges. That makes for very fertile soil." The Ph.D experience has increased my desire for knowing as much as possible about computer science. Further, it helped me to understand how can I devote my life into a career.

For the moment, I have to thank many people for this dissertation coming into existence. At the very first, I must thank *Professor Peter Herrmann*, my main supervisor, for his guidance and many useful discussions of my research topic; I'd like to thank him for guiding me to think in a systematic and global view in the engineering of software systems; I also appreciate that he has given me freedom for choosing my interested directions during the research. Appreciation also goes to my co-supervisor *Professor Rolv Bræk*, for his patience and discussion on writing papers with a more general view in the research domain. Looking back to years earlier, when I was a master student and working at SINTEF IKT in Oslo, my thankful gratitude goes to Professor, Chief scientist *Arne J. Berre*, who supervised my master work and research. He led me to the academic domain, which opened a gorgeous model-driven world to me, and he also showed his concerns for my Ph.D study during the last few years by E-mail contact. The three people influenced my career during the past seven years, and also will influence my future.

The submission of the dissertation is also at the same time when my daughter Mona Eryue was born to this planet. I would like to express my thankfulness for the almighty Lord to give me such a precious creature to my life.

My wife, Weiwei, thank you for your support such that I could concentrate on the research work and thank you for your company such that i don't feel lonely while staying in Trondheim. To my beloved

daughter, Eryue, Mona, hope you and your spirit will grow happily from the moment you land on this planet. Hope you have patience, kindness in your personality, and have a great desire for new knowledge.

Moreover, I like to thank *Professor Heinz Schmidt at RMIT and Professor Jan Olaf Blech at Aalto University in Finland* for their cooperation in the last few of publications of my P.h.D work.

Although our capabilities and technology have been expending geometrically, unfortunately, our ability to model their long-term behavior, which has also been increasing, has been increasing only arithmetically.

- Edward Tenner

Contents

List of Figures

List of Tables

List of Papers

Publications Included in the Thesis

- PAPER A:
  Fenglin Han, Surya Bahadur Kathayat, Hien Le, Rolv Braek and Peter Herrmann. *Towards Choreography Model Transformation via Graph Transformation.* Proceedings of the 2nd IEEE International Conference on Software Engineering and Service Sciences (ICSESS 2011). Beijing, July 15-17, 2011.

- PAPER B:
  Fenglin Han and Peter Herrmann. *Remedy of Mixed Initiative Conflicts in Model-based System Engineering*. Proceedings of the 11th International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2012), Volume 47 of the Electronic Communications of the EASST, 2012. Tallinn, 2012.

- PAPER C:
  Fenglin Han, Peter Herrmann and Hien Le. *Modeling and Verifying Real-time Properties of Reactive Systems*. Proceedings of the 18th International Conference on Engineering of Complex Computer Systems (ICECCS 2013). Singapore, 2013.

- PAPER D:
  Fenglin Han and Peter Herrmann. *Modeling Real-Time System Performance with Respect to Scheduling Analysis*. Proceedings of the 2013 International Joint Conference on Awareness Science and Technology and Ubi-Media Computing (iCAST 2013 & UMEDIA 2013). Aizu city, Japan, 2013.

- PAPER E:
  Fenglin Han, Jan Olaf Blech, Peter Herrmann and Heinz Schmidt. *Towards Verifying Safety Properties of Real-Time Probabilistic Systems*. Proceedings of the 11th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA 2014), Electronic Proceedings in Theoretical Computer Science 147, Grenoble, France. April 12, 2014.

- PAPER F:
  Peter Herrmann , Jan Olaf Blech, Fenglin Han and Heinz Schmidt. *A Model-based Toolchain to Verify Spatial Behavior of Cyber-Physical Systems*. International Journal of Web Services Research (IJWSR), 13(2016)1, pages 40-52, IGI Global.

  A previous version of this paper was awarded the Best-Track Paper Award of the Special Track on Reliability Technologies and Tools for Services-Based Systems at the 2014 Asia-Pacific Services Computing Conference (APSCC), Dec. 4-6, 2014, Fuzhou, China.

- PAPER G:
  Fenglin Han , Peter Herrmann , Jan Olaf Blech and Heinz Schmidt,Model-based Engineering and Analysis of Space-aware Systems Communicating via IEEE 802.11. *39th Annual International Computers, Software & Applications Conference (COMPSAC 2015).* pages 638-646, Taichung, Taiwan, July 2015, IEEE Computer.

I

OVERVIEW

1 INTRODUCTION

In recent years, software developers have observed the growing importance of applying Model-Driven Development (MDD) to software projects, MDD takes models as the most important artifact in software development which better reflects the problem structure and better solves the problems of software reuse and decomposition [156]. With recent advances in hardware and network technologies, software systems have grown to provide pervasive services to humans and society. A software system usually operates on a distributed and embedded computing environment consisting of diverse devices. The growing complexity of software systems requires a systematic engineering method. MDD promises to cope with the complexity of software development and introducing more automation in the process [185]. MDD has promised to be especially well-suited for the development of complex, heterogeneous, and large software systems [93]. Not only do complex systems rely on models as an important planning, structuring and development tool, but models can also provide measurability and evolvability for complex systems.

In order to provide modeling of different levels of abstraction and introduce automation to software development, we introduced the Reactive Blocks approach to Complex System Engineering (CSE) since the Reactive Blocks approach solves a number of CSE problems including model evolution, abstraction, automatic code generation, and tracing.

The basis of our work is the engineering technique Reactive Blocks that is introduced in Sect. 1.1. A particular property of Reactive Blocks is its ability to specify and verify software flows which we discuss in Sect. 1.2. An important solution for the verification of systems is model-based analysis which is described in Sect. 1.3. In Sect. 1.4., we sketch how formal methods are used in Reactive Blocks. For particular importance of our work is the modelling and analysis of real-time systems that we discuss shortly in Sect. 1.5. Finally, we present our research methodology in Sect. 1.6 and the research question leading our Ph.D. project in Sect. 1.7.

1 The Reactive Blocks Approach for Complex System Engineering

The Reactive Blocks approach (formerly called Arctis) [110, 118] is a method aiming at developing reactive distributed software systems. It maintains a library of blocks with maximized modularity and reusability (on average about 70% of the code of a system can be reused from the library, see [114, 116]). The system behaviors in Reactive Blocks are modelled with UML activity diagrams. An Activity can be hierarchically composed by others activities. Interactions between activities are modelled by UML pins and flows through which data or control signals can be transferred from one activity to the other. The visible behaviour of a decomposed activity (called a block) is modelled by contracts in the form of so-called External State Machines (ESMs, [114]). The activities have formal semantics based on their token-flow nature (similar to Petri-nets) [115, 151], the activities are facilitated with formal semantics [115], which enables verification of relevant properties, e.g., the correct integration of building blocks into activities, by the model checker included in the tool-support Arctis [118].

A good example of Reactive Blocks and the advantages for combing Reactive Blocks with a popular industrial architecture will be given in Chapter *2*, in which we will also give a more detailed introduction of Reactive Blocks.

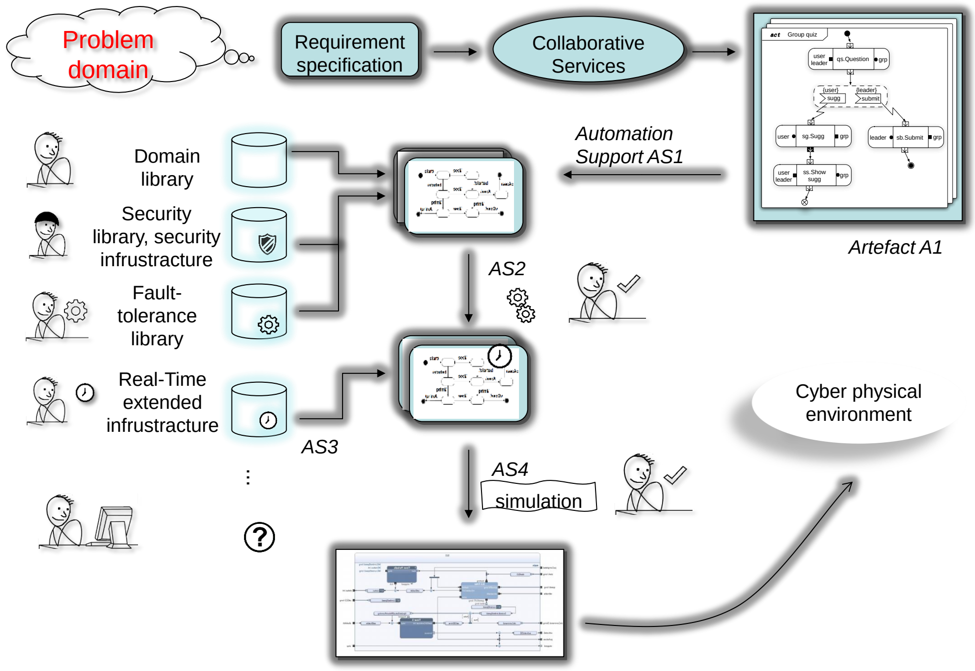

Problem domain
Requirement specification
Collaborative Services
Artefact A1
Automation Support AS1
Domain library
Security library, security infrustracture
Fault-tolerance library
Real-Time extended infrustracture
AS2
AS3
AS4
simulation
Cyber physical environment

Figure 1.1. Reactive Blocks approach for Complex System Engineering

Model driven development (MDD) is seen as a way to handle complex systems. The model-based approach shows its advantage in describing domain specific problems. We summarise the work steps for model-based CSE into a procedure that corresponds to the core work steps of MDD. The core of MDD is a series of activities including requirements modelling, functional modelling, model based development and testing, code generation, system synthesis and system generation. In our procedure, we solve domain specific problems in terms of MDD. Additional properties need to be measured and evaluated, e.g., system performance, real-time limitations, system safety and robustness.

The extended steps for CSE require extended MDD approaches and facilities to support the system development according to the end-user requirement. We designed a development process shown in Figure 1.1 and described the Reactive Blocks approach for CSE:

- System requirements are initially described by abstract models that capture main activities or artifacts in a user-centric manner. The requirements models primarily contain collaborative service models and UML activities. System level behaviour are described by so called choreography models (Artifact 1 in Figure 1.1). Since Reactive Blocks are used as design time modelling technique, a more abstract model is needed for capturing functional specifications.

- The system level behaviour can be mapped into Reactive Blocks models, and functional specifications are captured by such decomposition analysis. In this step, we introduce the rule-based model transformation for automation support (AS_1 in Figure 1.1).

- Functional components are described as building blocks using reactive blocks. Existing functional blocks can be reused from a library where we maintain a repository of domain specific building blocks (Domain library in Figure 1.1).

- In the following steps, non-functional properties of specific functional building blocks are analysed and relevant complement mechanisms are amended to the building blocks. For example, fault-tolerance mechanisms can be amended to the building blocks by formal analysis based on built-in fault detection mechanisms [182] (fault-tolerance library in Figure 1.1); Reactive blocks also maintain a security library and built-in security mechanisms, such as encryption and integrity measures to protect the system against untrusted communication channels or malicious attacks [81] (Security library and security infrastructure in Figure 1.1).

- We preserve a transformation rule library to support the automatic remedy of typical error patterns on distributed concurrent systems. This aids the automation support for system fault detection and repair (Automation Support AS_2 in Figure 1.1).
- For more sophisticated requirements at system level like soft real-time (implemented by software) or safety requirements (related to probabilistic real-time requirements), we analyse the synthesized system and abstract the real-time behavior into corresponding analysis artifacts. For example, augmented real-time annotations and building blocks can be transformed into timed automata for real-time properties. In [179], Slåtten addressed fault tolerance mechanisms for reactive embedded systems to improve the system reliability and avoid side effect. We extended the system to be able to model safety properties such that reliability is extended to a cyber-physical context.

  For these kinds of analysis capabilities we imported the real-time extended infrastructure, which contains primarily real-time analysis tools such as UPPAAL and PRISM. This kind of analysis and infrastructure is supported with automatic model transformation and automatic generation of reasoning formulas (Automation Support AS_3 in Figure 1.1).
- Real-time related properties of embedded systems are hard to verify since they encompass extremely tight integration of and coordination between information world and physical resources. Because of the uncertainty and uncontrollable conditions of the physical world [98], we have to compromise with respect to the verification of real-time software systems in a cyber-physical environment. To carry out such kind of research, we analyse the real-time related properties of autonomous systems in concrete scenarios and strictly defined physical environments. A case study is an automatic transportation robot running under a remote control system which are coordinated using 802.11 wireless protocols. We imported automation support in this phase for supporting the translation of software models into the verification language and automatic generation of a proof theorem (automation support AS_4 in Figure 1.1).

In the functional development phases and simulation in the cyber-physical environment as shown in the central part of of Figure 1.1, analysis work is done together with development work in a model-centric manner. We use the model as an essential abstraction of functional behaviour and apply analysis tools to carry out the analysis work.

By taking advantage of the graph nature of Reactive Blocks and importing the model transformation, our approach is flexible and extensible for various properties' analysis. When applying system level analysis, we first have a functional correctness preserved building block for input to a system level property analysis. Such analysis usually needs to take advantage of the graph nature of the building block, make a mapping to the target analysis language, and then transform the model to the target language. The model transformation can assist in providing automation in this procedure. The analysis result from the target language is feedback to the reactive blocks describing whether or to what extent the building block has fulfilled the required system level property. The ⑦ in Figure 1.1 represents potential possibilities like fuzz testing [175] or machine learning based software fault detection [145].

The main contribution of this thesis is marked as *AS1* to *AS4* as indicated in Figure 1.1. They represent: *AS1*: Automation support for transforming requirement to design time model; *AS2*: Automation support for system fault detection and remedy; *AS3*: Automation support for real-time behavior modeling and verification, especially expressing time constraints on safety-critical systems; *AS4*: Automation support for safety analysis of cyber-physical systems. The contributions with its corresponding analysis work are presented in seven papers and attached to the end of this thesis.

MDD has been a research hot-spot for many years, and several challenges have already been solved while many others are still open [185]. The challenges include many aspects including requirement modeling, domain-specific modeling, model verification and validation, etc. The Reactive Blocks approach is a built upon the diligent work of many researchers. A detailed summary of the research works contributed to Reactive Blocks approach by others is summarized in Chapter *6* Sect 1.

We will talk in detail about the contribution of this thesis to the Reactive Blocks approach for Model-driven development in the rest of this chapter and in Chapter *6*.

2 Specification and Verification in Reactive Blocks

In this section, we review four aspects of the SPACE specification method [110] on which Reactive Blocks is based: *structural*, *transformational*, *compositional* and *analytical*. We briefly mention our contributions to the four specification aspects in this thesis and their novelty.

- **Structural:** This aspect is concerned with the description of distributed reactive systems. From a software point of view, this relates to how the software system organizes its codes and how the software modules map to the topology of the distributed system, including its processes, components, and connections. The SPACE method provides *Reactive Blocks* as basic specification and composition units for reactive systems. In a *building block*, system behavior is modeled by UML 2.0 activities that, in a Petri net manner, express behavior by tokens flowing via the edges of a graph. Also, an activity may contain a call behavior action referring to another activity. Such structures can be reused hierarchically and compositionally in a way such that each building block composes a part of the whole system graph. The partial system described by a building block is translated into a set of state machines, the possible execution of an event system can be analysed and simulated, by extends its *state space*, see [115]. This thesis extends the structure of the SPACE specification method by new annotations and labels helping to extend the state space of a reactive block. The annotations and labels, which contain count down clocks and guide conditions, allows us also to model QoS properties such as real-time constraints and probabilistic behaviors. In effect we extend the semantics of Reactive Blocks such that it is possible to use other analysis tools.

- **Transformational:** A central feature of the SPACE method is the transformation of UML activities into petri-net based state-transition graphs, such that, the UML activity diagram becomes a software model with formal semantics [113]. In this thesis, we contribute to the SPACE method by applying model transformation in two ways. At the beginning, we introduce a rule-based model transformation approach for model decomposition and pattern recognition. We describe the system patterns using sub-graphs and apply model transformation to the recognized pattern for certain purposes including system derivation or erroneous pattern remedies. Both the pattern and the transformation are formatted in graph transformation rules. A graph transformation rule contains a pre-graph and a post-graph, in which the former describes a recognition and the latter makes a mark or evolution. The graph transformation rules and their compositions for transformation enhance the SPACE method so that more automated software development can be achieved.

  Then, we extend the transformation into general purpose: use model transformation to transform an extended structure of a reactive block to formal languages. We transform the annotated reactive block with its state space into timed automata (see Paper C) and probabilistic timed automata (see Paper E), the annotations are maintained in a performance profile (see Paper D). The transformations consider both, the Reactive Blocks' state space, which is generated by the Reactive Blocks tool-set, and the additional annotations, that are attached to the blocks by plugins. This automatic transformation can generate models in analysis languages like the query language in UPPAAL, which is a subset of CTL (computation tree logic) [87, 6]. Alternatively, we can create models in the PRISM language [120], which supports a rich set of probabilistic models, and we apply the analysis based on Probabilistic Timed automata (PTAs, see [102]).

- **Compositional:** That is another fundamental aspect of modeling concurrent distributed software systems. The compositional aspect concerns how the software system is composed and how the systems is finally derived. The SPACE method uses an External State Machine (ESM, [114]) for system composition, and the ESM performs three key functions: First, an ESM is an abstraction

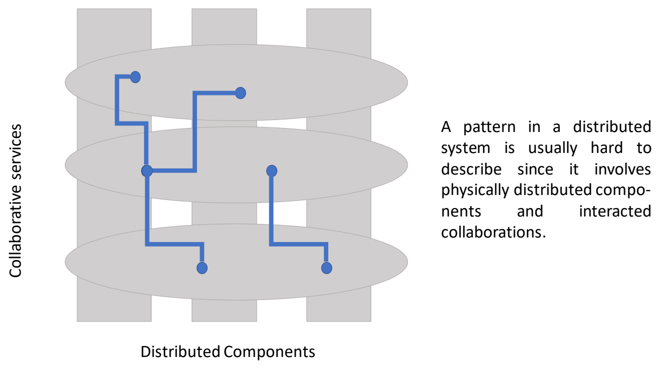

Collaborative services
Distributed Components
A pattern in a distributed system is usually hard to describe since it involves physically distributed components and interacted collaborations.

Figure 1.2. A system pattern that acquire analysis techniques.

of building block behaviour and also a communication interface between a building block and its environment; second, the ESM contract strictly defines the inputs and outputs of each block and their causality; third, large building blocks are composed using sub-blocks and UML activities such that the ESM becomes the composition contract between internal and external blocks. In this thesis, we first align such verification techniques to rule based analysis. Through model verification, we discover erroneous patterns to which we can apply automatic system detection and correction. Model transformation rules can automatically compose domain specific blocks to solve an erroneous scenario identified by graph patterns and verification results. Second, we defined several new extensions of ESMs to produce the Real-Time External State Machine (RTESM) and Probabilistic Real-Time External State Machine (PRTESM) for more building block behavior formalization and verification. RTESM and PRTESM define real-time and probabilistic real-time behavior of the system. Thus, our specification methods are enriched to describe more complicated system level behaviors in a compositional manner.

- **Analytical:** The analysis ability is a highly desirable feature and focus of research over the past decades for distributed software systems. This is due to the increasing complexity and chaos of large software systems. Since reactive systems can have parallel or concurrent behaviors, the correctness of the ESM becomes very important. In particular, the ESM defines the interaction of internal and external behaviors around a building block. SPACE applies the compositional Temporal Logic of Actions (cTLA, see [95, 94]) for the semantics of the ESM. The analysis and verification techniques in SPACE is based primarily on cTLA [111], which is a compositional variant of Lamport's Temporal Logic of Actions (TLA) [127]. In this thesis, we provide the SPACE method with an extended ESM and apply formal verification also for real-time and probabilistic behavior by importing timed automata and computation tree logic, including timed computation tree logic (TCTL) and probabilistic TCTL (PTCTL). We enable the compositional verification of real-time (probabilistic) behavior by flattening the hierarchical structure of building blocks to a network of timed automata.

To summarize, this thesis utilizes the graph nature of the Reactive Blocks approach and introduces model transformation as an artifact for automation in designing reactive systems. Also, the model transformation supports extended modeling capabilities so that real-time and probabilistic behaviors can also be captured. Based on the extended modeling capabilities, we can formally prove safety and spatiotemporal properties of some real-time and safety-critical systems. More details on the contributions of this thesis work are presented in Chapter 6.

3 Approaches to Model-Based Analysis

Component analysis is a central activity in software engineering. Many approaches have been investigated and published by researchers and industrial practitioners (see Chapter *6*). In our model-based approach for software component analysis, we pursue a path that can customize fundamental theories and facilities to the needs of reactive systems. Due to the common structure of software systems, which includes concurrent process communication and a state-transition-based model specified as a graph in Reactive Blocks, we choose an approach that specifies system components by modeling local behaviors in functionally separated building blocks. The composed system can be decomposed by its functional nature and some common types of system properties can be analyzed and even automatically corrected.

Figure 1.2 shows a system pattern that requires analysis. We can evaluate existing Reactive Block models and discover the cross-cutting nature of erroneous scenarios of a reactive system. Traditional analysis techniques which include most software testing approaches, are usually simulation based. Our approach combines formal verification for error discovery with graph transformation for correction. The patterns relate to erroneous behavior and to graph transformation rules that are combined to analysis patterns [70]. The analysis patterns can be used for various Reactive Blocks-based models.

Other purposes of analysis include testing system performance under certain loads or evaluating system delay. We combine this type of simulation-based analysis with our approach at the end of Chapter **4**. To summarize, the analysis of a component-centered software system, under our model-driven method, includes the following purposes:

Pattern Discovery

A common definition for a pattern is a solution to a problem in a context [72]. In our approach, we discover two types of patterns: one pertaining to software system derivation and decomposition, and the other is common bug patterns due to the nature of reactive distributed systems. We formalize the two patterns using graph transformation rules.

Restructuring

Chikofsky and Cross define *refactoring* as follows: *restructuring is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior (functionality and semantics)* [46].

For the erroneous pattern we discovered, we are able to apply automatic refactoring using graph transformation rules. This is achieved by incrementally applying a set of graph grammar rules. Such transformations maintain the old system behavior while importing new behaviors that can remedy the erroneous pattern.

Figure 1.3 shows the architecture of applying model transformation rules to building blocks for automatic system remedy. In the transformation, we preserve the semantics of the building blocks, and the transformation rules are divided into different phases. On the left, early rules detect inconsistencies and add labels, so as to form the pattern for later phase rules for operations and remedy. On the right, new behaviors are added to systems such that the functional behavior is not altered, but remedy logic is inserted into the system. The case studied in this approach is a mixed initiative conflict problem that commonly occurs in distributed reactive systems. The overall approach provides a quick fix and automatic remedy for SPACE domain specific modeling . On the top, we use UML meta-models to carry out the transformation, thus preserving the correctness of the transformation and its production.

CPS and Networked Control System

Cyber-physical systems (CPS) impose great need for system level property analysis. One typical CPS is the networked control systems that poses several challenges related to time and event-driven computing, software, variable time delays, and failures [161]. One important cornerstone for the analysis of CPS is a modular design and development of cyber-physical systems. Based on the Reactive Blocks approach, we reinforce the models with labeled transformations and annotated variables which provide an architecture for CPS.

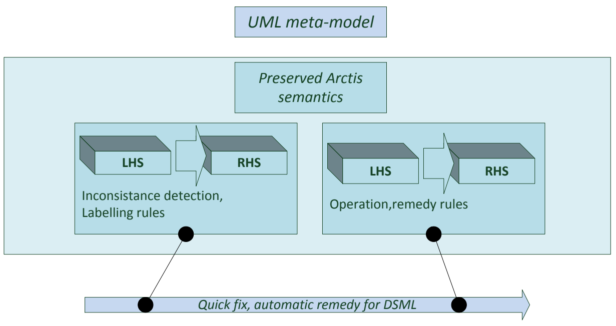

UML meta-model
Preserved Arctis semantics
LHS
RHS
Inconsistance detection, Labelling rules
LHS
RHS
Operation,remedy rules
Quick fix, automatic remedy for DSML

Figure 1.3. Graph Transformation Rule based automatic System analysis and Remedy

New Formalism and Domain Specific Requirements

Reactive systems contain more specific features that need new formalism support and verification. Especially when the reactive systems constantly contact with environmental resources and exchange information. The vast growing application of cyber-physical systems presents new analysis needs for properties like reliability, safety and security [169]. Real-time formalization and analysis is a critical property related with safety and performance of CPS. Thus, there is a need to extend reactive blocks to formalize and analyze real-time related properties. This thesis gives special attention to real-time issues for reactive systems modelling, analysis and verification. We extended the external visible behaviour abstract (ESMs) to present abstracted real-time and probabilistic behaviors. With extended system abstraction, we are able to carry out real-time and probabilistic analysis for reactive systems.

4 Formal Methods Applied in Reactive Blocks

In this section, we briefly summarize the formal verification applied to Reactive Blocks. Model checking is a technique applied to finite-state concurrent systems, where a specification about the system is expressed by temporal logic formulas, and efficient algorithms are used to traverse the model [1] defined by the system and check if a specification holds or not. Amir Pnueli ([159]) and Leslie Lamport ([125]) introduced temporal logic to strictly define the semantics of software systems. Large complex and seemingly chaotic system behaviors have become meaningful and understandable (see [4]). Amir Pnueli points out that "*We should concentrate on an engineering approach assembling many tools and methods among which* ***Formal Verification*** *should be a major player*". Therefore, this thesis intends to expend Reactive Blocks approach in this direction.

The introduction of additional formal methods to the Reactive Blocks approach gives building blocks a number of significant benefits. For example, since the Reactive Blocks approach calls for building a reusable library of software components, the formal method can guarantee the building blocks are free of certain anomalies. Moreover, formal methods can prove the building blocks will behave as they are designed. Also, formal methods can check the appropriateness of the building blocks for their environment if the environment is fully defined. Once the verification work is done for a building block, the block is universal reusable everywhere [95].

Finally, the model checking introduced in Reactive Blocks allows for automatically generating formal specifications in TLA+ [127] and certain theorems that can verify the common properties of reactive systems such as dead-lock and infinite loops [177]. Thus the formal reasoning discharges interactive proofs that requires deeper understanding of theorem proving.

[1]This model definition is different from the model concept we commonly used in this dissertation. Here, the model is the specification language that abstract certain building block behavior and input to a verification tool.

Formal reasoning is often highly complex and can usually only be carried out by specialists; thus, it is expensive [83]. Also, formal reasoning needs to be done if the properties needs to be defined manually or with interactive tools like certain theorem provers. By creating reusable libraries of building blocks, we separate the complex work of formal reasoning with common development work [95]. We also avoid the increased project expense stemming from the application of formal verification since the reusable building blocks are reusable everywhere once they are created.

Basic Semantics of Reactive Blocks

The Reactive Blocks approach takes the UML activities as the artifact for modelling of system behaviours, and such behaviors are translated directly into executable state machines [113]. In [111], Kraemer and Herrmann formalized the precise semantics of Reactive Blocks which gives the foundation for model checking an automatic synthesis of component-based implementation. The Semantics of Reactive Blocks is built upon cTLA [95] which is an extension of Leslie Lamport's Temporal Logic of Actions (TLA, [127]). Compositional cTLA processes specify systems and subsystems as composition of simple cTLA process instances which cooperate by means of synchronously executed process actions. The transformational mapping of the behavior from the activities to that of the state machines is proved by refinement mappings [2].

With the formalization, a specification is automatically transformed into TLA+ and theorem on specification can be proved automatically by the TLC checker [201].

Normally, formal verification needs the following working steps to support the formal reasoning of a software system:

- Transformation into a formal language: The system implementation needs to be abstracted and specified using a formal language that is feasible for specifying a theorem and the language should be a refinement of the original implementation. In Reactive Blocks, the original activity based system model is translated into TLA+ formula using inbuilt formulator [177].
- Using the formal checker to verify theorems against specification: In a second step, the system properties are specified using formal language. Thus formal checkers can be used to verify the specification against the theorems. If no theorem is violated, the analysis ends successfully. Otherwise, the formal checker provides a counterexample in which a formula is violated.
- Post processing for tracing of counterexamples: If a counterexample is generated, a trace is usually presented on the original specification to show the errors such that the developers can be assisted to correct the error and thus such processes is repeated until a correct model is finally achieved. In the assisting tool, Reactive Blocks, a diagnosis is provided for a violated theorem and in some cases, automatic fix can be provided. This is done by Slåtten in [178].

Many language and verification tools are designed for formal reasoning with different areas of emphasis, for example, UPPAAL [19] supporting timed automata is the tool for real-time verification with Timed Computational Tree Logic (TCTL) [130, 6]; PRISM [120] is the tool for probabilistic symbolic model checking based on Probabilistic Computational Tree Logic (PCTL) [120]. More tools and approaches can be found in [59, 195].

Given these tools, we propose an approach that extends formal verification of Reactive Blocks with the supporting power of model transformation. The extensions is targeted at the application of Reactive Blocks to more complex systems development in cyber-physical system that needs to consider real-time and probabilistic behaviors of systems and of the systems that are affected by communication protocols and interactions with environment.

5 Real-Time System Modeling and Analysis

Real-time Systems are complex and error-prone since they need additional mechanisms to support timed behaviors. Real-time reactive systems are a set of software systems in which correct functioning depends not only on values of the results produced but also on the physical time within which the results are produced. Also the time constraints can be varied according to the strictness of the time requirement.

A *hard real-time system* requires the time limit to be **always** met, while a *soft real-time system* needs the time limit to be **often** met [41]. Important application domains for real-time systems include, for example, embedded systems including micro-controllers that are connected to complete systems via sensors, actors, operator controls, and communication devices.

Traditionally the verification of control software for hard real-time systems is based on low level approaches, such as programmable timers, direct handling of devices, and assembly languages which aim at optimizing predictable execution time of code on embedded systems. The drawbacks of these approaches are obvious in that the verification results do not give an over view of the whole system in that aspect. We aim to enable real-time system analysis at the modeling level, then the system characteristics, such as performance and real-time constraints, can be analyzed in a structural view and from the top level.

6 Research Method

The main research challenge resides in the modeling and design of transformations for domain-specific models. A constrained case-study based research method is applied throughout each research phase; that is, we carry out case studies in applying model transformation and verification in concrete scenarios of model-based reactive system development. In the initial phase, a detailed list of possible domain-specific features for distributed reactive systems must be surveyed. Among the possible features, a careful selection of features is studied and modeled to see what problems can be solved by model transformation and what analysis can be carried out. So the research is designed in several phases, including survey, modeling, transformation, analysis, and verification. Case studies are the central work during the research. During each case study, the survey, modeling, transformation, analysis, and verification phrases are carried out in a spiral manner:

- Survey: That includes massive paper reading including the survey of domain features of distributed systems, modeling capabilities for such features, transformation method, as well as the analysis and verification method.
- Modeling: After picking up candidate features, we first need to give a domain specific model for the software system and discover the dynamic aspects of the model that can be captured in the transformation model. Since our research method is centered on software verification, we also need to consider what formal properties, which are usually expressed as temporal logic formulas, can be captured in the new formalism, and whether such formal properties can be applied to formal verification to support our modeling and transformation purpose.
- Transformation: With the given model and the chosen transformation method, the transformation experiment is carried out to test the designed model and transformation rules.
- Analysis and verification: After transforming the model into a formal language that can be accepted by domain-specific analysis tools, we carry out typical analysis and verification of our models. During this phase, the model also needs to be revised to see whether the problem pattern has been correctly modeled and whether the analysis and verification result is to our expectation.

7 Research Questions

In a review and forecast for software engineering in the 20^{th} and 21^{st} centuries [32], in 2020s and beyond, with the increasing computational plenty, the software engineering tools should be able to provide more capabilities in specifying behaviors, generating resulting applications, verifying and validating capabilities, performance and dependabilities, and integrating them into even more complex systems of systems. With such richness in mind, we need to expend the Reactive Blocks approach in dimension and depth, both in academic and engineering practice. With the firm established foundation for the Reactive Blocks approach for visual programming of concurrent reactive systems, challenges and opportunities coexists in the research work. I researched in different directions and promoted research questions that will be discussed in the following:

- **RQ.1** *How can model transformation be used in a specification language? What purpose can it achieve?* (corresponds to automation support *AS1* in figure 1.1)

  At the beginning of the research work, model transformation was the main approach we considered for the SPACE method. A list of surveyed features in automatic software engineering has been found for the application of model transformation. We picked up typical application cases, such as model decomposition and refinement and automatic error remedies in software models.

We have the Reactive Blocks method and tool-set as a foundation aiming at extending the model-driven compositional development of reactive systems to the more complicated scenarios. During the process, more questions with the modeling capabilities and formal back-end that meet the challenge of new complex application scenarios. Specially, when considering cyber-physical systems, Reactive Blocks needs to support, e.g., time and performance issues. We arise a more general question with respect to extending Reactive Blocks:

- **RQ.2** *How can Reactive Blocks sufficiently support the modeling and formal back-end of real-time and performance issues?* (corresponds to automation support *AS2* and *AS3* in figure 1.1)

  This research question can be further divided into three sub-questions:

  SubRQ.2.1 *How can Reactive Blocks support the compositional modelling of real-time properties for reactive systems?*

  SubRQ.2.2 *How can formal reasoning for timed properties be appended to Reactive Blocks?*

  SubRQ.2.3 *Since Reactive Blocks is closely related to networked systems, how can we integrate a mechanism to measure the performance of Reactive Blocks?*

Following the research of model-based reactive engineering, we seek to extend the modeling capabilities of the Reactive Blocks method.

Since we aim to extend the Reactive Blocks approach for CSE and more domains like cyber-physical systems, real-time systems. We not only consider the formal aspects of complex system, but also the non-trivial aspects like how the new appended mechanisms can contribute some non-functional aspects of complex system engineering. We ask the following question:

- **RQ.3** *Can those extended capabilities be used in emerging needs of software engineering? For instance, how can it help to improve the reliability or robustness of the software?* (corresponds to automation support *AS4* in figure 1.1)

After the research, we need to find a practical approach for synthesising the Reactive Block approach to common software engineering practices. We describe a feasible architecture that applies industrial practices of Reactive Blocks with common API programming in Chapter *2*.

2 MODELING

In this chapter, we will have a brief introduction on the two levels of models we applied in our research. First, is the choreography model for service engineering; then is the reactive building blocks, its basic concepts, composition and application in a IoT system scenario.

1 Choreography Model

Choreography model [66] is used by service engineers to capture behaviours in the collaboration of distributed reactive systems. In a choreography model, collaborations can be defined by UML activities connecting actions by flows. Actions may either specify the behaviour of a collaboration or a local activity. Collaborative actions contain references to their participants in the form of roles. Flows may contain intermediate control nodes (e.g. start, stop, choice, merge, fork and join) defining the ordering and causality among the actions. We take a *Train Control Service* scenario as an example and show how choreography can be used to model the global behavior of the services with its sub-collaborations.

The TrainControlService system has two main participants,Train and radio block center (RBC), represented as roles. The train and the RBC participate in three collaborative sub-services: PositionReport, MoveAuthority, and HandoverSupervision which perform the following activities:

- *PositionReport* reports the current train position to the RBC.
- *MoveAuthority* sends the safe travel distance from the RBC to the train.
- *HandoverSupervision* transfers the train control supervision to the new RBC if the train travels to a different region.

Feature 2.1 shows the global behaviour of a train control service using a choreography model. A train on its journey reports its current position in intervals to the RBC which is responsible for the region the train operates in. This operation is specified by the collaboration PositionReport. Thereafter, the RBC validates the received position information of the train via the local activity SupervisionLogic. If the information about the location of the train is correct, the RBC issues successive movement authorities (MA) to the train which is modeled by the collaboration MoveAuthority. Finally, if the train crosses the border between two regions, the collaboration HandoverSupervision is invoked. There flows between the four collaborations show the casual relationships of the activity and common syntax like the merge node, fork and decision nodes, are also applied in the diagram to connect the flows. Merge node *M1* joins the casual relations from collaborations *MoveAuthority* and *handoverSupervision* to *PositionReport*, while decision node *D1* signifies that either *MoveAuthority* authorises the hand over or a flow final of the current activity.

The *Train Control Service* choreography model is a global behaviour that can be captured by service engineers as close to the problem domains as possible. In our research, we applied graph transformation rules to it and contribute to how can we add automation to the generalisation from service models to a step further down to actual component definition.

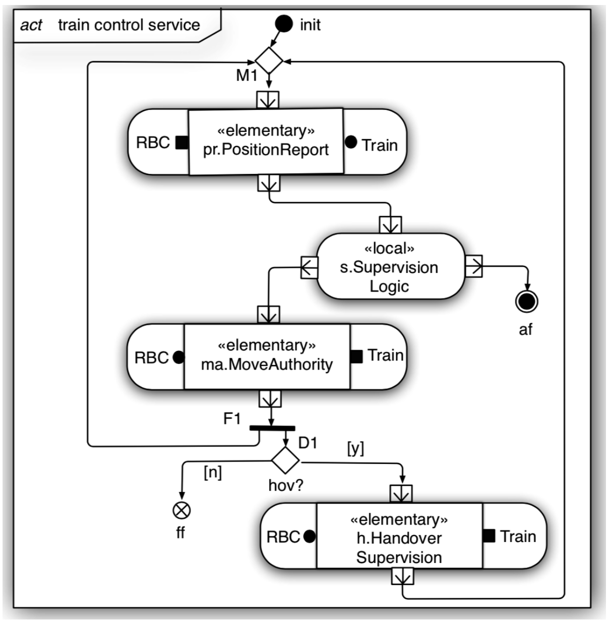


Figure 2.1. The train control service global choreography model: the choreography model models collaborative services using UML activities connecting actions and flows. The diagram contains three collaborative services of the two participants.

2 Reactive Blocks

As the number of sensor devices and actuators grows and billions of devices need to connect to the internet, an emerging large complex networked software system connecting physical world and internet, which is known as the Internet of Things (IoT), is expending geometrically. IoT is on track to connect 50 billion “smart” things by 2020, and forecasts say that the number of the IoT devices will reach 1 trillion sensors soon after. An IoT system aggregates sensor data, translates between sensors protocols, process sensor data and usually contains multiple programming APIs and communication protocols. Usually the protocol programming occupies the most of an IoT system. This part will give an example of Reactive Blocks for a protocol block picked up from the public Bitreactive[1] repository and its applications scenario in a IoT solution.

Reactive Blocks is an approach that synthesizes the adapted UML activity diagram as its representation syntax, encapsulate model transformation, code generation and formal methods as its supporting back-end, with a full stack tool developed in the Eclipse environment, and professional for building reactive concurrent systems. Bitreactive AS is a company established based on the Reactive Blocks paradigm for providing solutions for IoT industry; It provides a global repository of highly reusable and robust software components aiming at shorten the development time for IoT solutions.

3 Reactive Blocks for encapsulating IoT protocols

The most benefits of Reactive Blocks for IoT system development is to shorten the learning time for uses of new protocols and to maximise the reuse of code, while at the same time to guarantee that the blocks can be reused in other Java based projects and interact with normal API-based programming. In this section, we will give an Advanced Message Queue Protocol (AMQP) example and its application in an IoT solution.

[1]Bitreactive AS: `http://bitreactive.com`

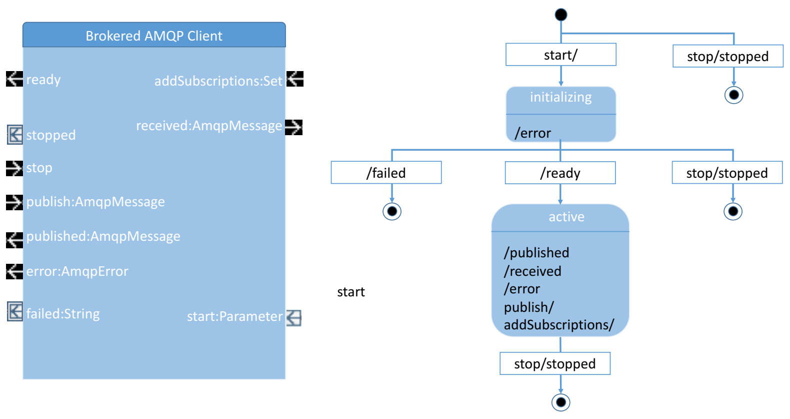


Figure 2.2. Advanced Messaging Queue Protocol (AMQP) from BitReactive

AMQP Libraries

The Advanced Message Queue Protocol (AMQP) protocol is an open standard application layer protocol for asynchronous message queuing. It permits almost any form of messaging, and Bitreactive provides an off-the-shelf building blocks library for AMQP. The AMQP library contains five building blocks supporting point-to-point messages via brokers. Figure 2.2 shows the *Brokered AMQP Client* block which can publish and receive AMQP messages to/from an AMQP broker. An AMQP broker can be embedded into any software component that sends or receives AMQP messages.

A building block is composed of two parts: The implementation is a UML activities diagram with parameter pins as shown on the left side of Figure 2.2, which encapsulates Java method calls inside via block references or action calls; and the External State Machine (ESM) as shown on right side of Figure 2.2. The UML activities (now shown on the picture) give the implementation of the behaviors of the block and only gives the input/output control or parameter flows. The ESM, which is also called the External Contract between the building block and the environment, defines the causality behavior of the block.

As the external contract shows, the *Brokered AMQP Client* block is started via a token arriving at the *start* pin. Thereafter, the component arrives at the *initializing* state. If the broker is connected to the AMQP broker server successfully, the block will go to the *active* state, and notifies the environment via the *ready* stream output pin, else it will send out a *AmqpError* object through the *error* pin after which it continues to stay in *initializing* state. If the connections fails, it will terminate the component via *failed* pin. The component can publish or receive AMQP messages via the *publish* and *received* parameter pins, when a message is published successfully, a notification will be given to the environment via the *published* streaming output pin [2]. Both the *start* and the *addSubscriptions* can be used for adding new broker addresses. They include the host address, port, and queue names. If we look into the *Brokered AMQP Client* block (Figure 2.3), it contains two blocks called *Brokered Receiver* and *Sender*, which separates the main send and receive functional codes. This hierarchical composed system guarantees that a block is functionally isolated, complete and reusable, and the blocks can be composed together to form a large powerful block. While at the same time the formal backend can guarantee the composition is functioning as it is designed and pitfall free for concurrent execution. The library contains all the needed functions for AMQP protocol and can be reused in any client for receiving, publishing AMQP messages, and add subscription addresses.

[2]A streaming output pin can pass data from a block when the block is active. More pin types can be found at `http://reference.bitreactive.com/reference/state-machine-blocks.html`

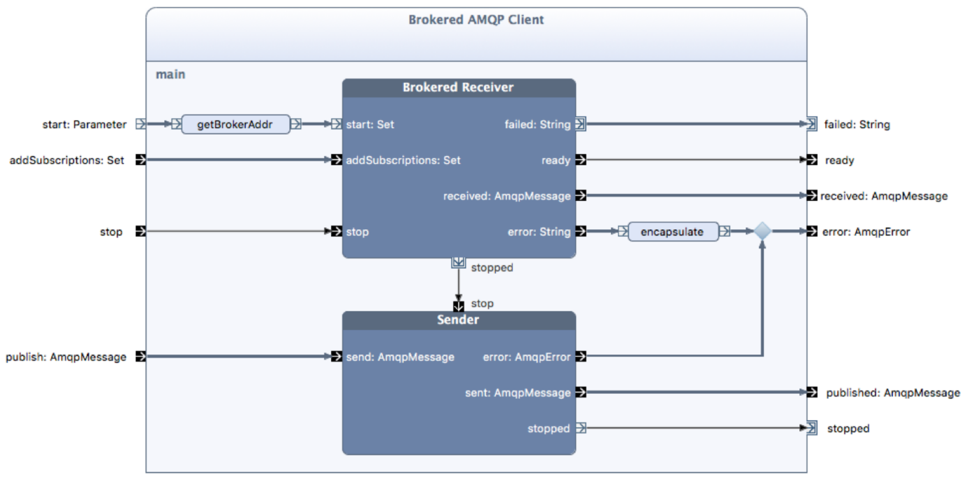


Figure 2.3. Brokered AMQP Client Activities

4 Architecture of IoT Solutions

Figure 2.4 describes an architecture where reactive building blocks can be massively reused in industrial solutions. In the delineated solution, an IoT system is a distributed hardware and software system, where sensors, controllers and actuators communicate with the local gateway and data aggregator, the sensors and controllers are the bottom layer hardwares that each other has a virtual representation (usually the gateways) in an IoT system. The hardware, on which the system runs, communicates with the IoT system using LAN and IoT protocols, e.g., Bluetooth Low Energy, ZigBee, LoRaWAN protocols, and the gateways are developed as a set of spiderwebs. The middleware (gateways, micro-components) can interact with external, internal using Restful APIs such that a wide area network can retrieve data from such systems. Below, we first describe the the home health care system, then give a core component example.

Figure 2.4 shows a architecture of an IoT solution which describes a home-care (HCS) IoT system scenario. The system needs to provide various sensor data, e.g., for the health, safety of habitats and monitoring of habitat environment. Further, the system has to offer various UIs including immutable terminals like PC, smart TV, mobile terminals like smart phones and smart wearable devices, web and remote monitoring UIs, e.g., for hospitals. The system should be able to encapsulate various protocols, e.g., Bluetooth Low Energy (BLE), ZigBee, LoRaWAN or many other proprietary wired protocols as its infrastructure. Thus a multitude of software components serving the various protocols can be envisaged. The complex set of interactions and their permutations leads to a diverse set of APIs.

A token distribution component in IoT

A key issue for IoT system is how to secure the communication and data access of the decoupled services. At present, a common way for security is by using a generated token [146] to securely share authentication information between services and clients. Such mechanisms can encode authentication information directly in the token instead of needing to be required from a central location on every service call. A common way for acquiring such tokens is the Single Sign-on (SSO) [184] technique, such that the authentication service can create a Json Web Token (JWT) using a private RSA key to sign the token, and this JWT is return to the client. The client can then use this JWT on subsequent calls, and the integrity of this token can be checked against the public RSA key of the authentication service.

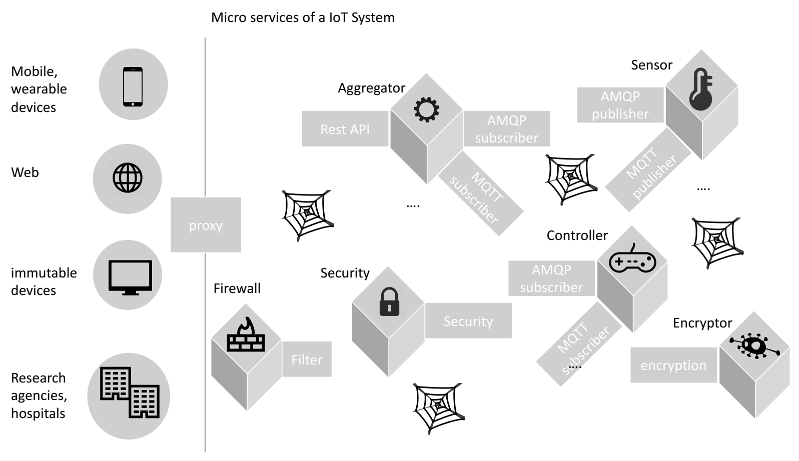


Figure 2.4. IoT architecture

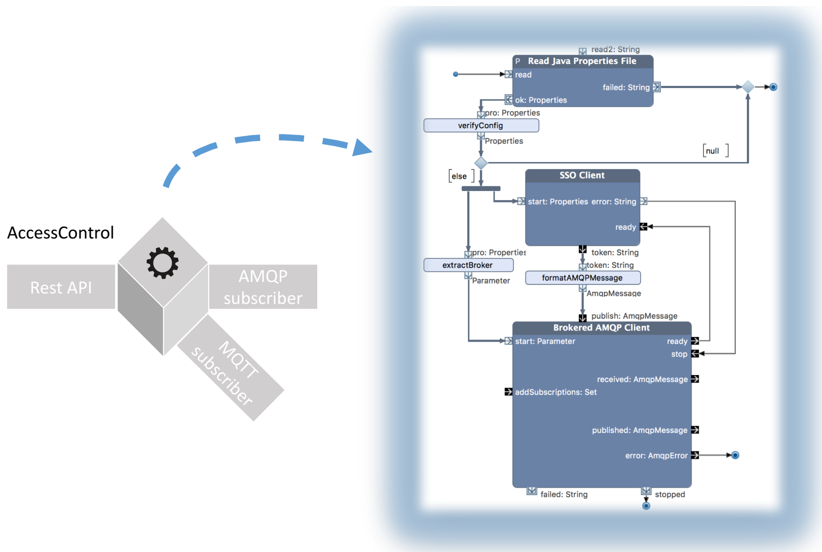
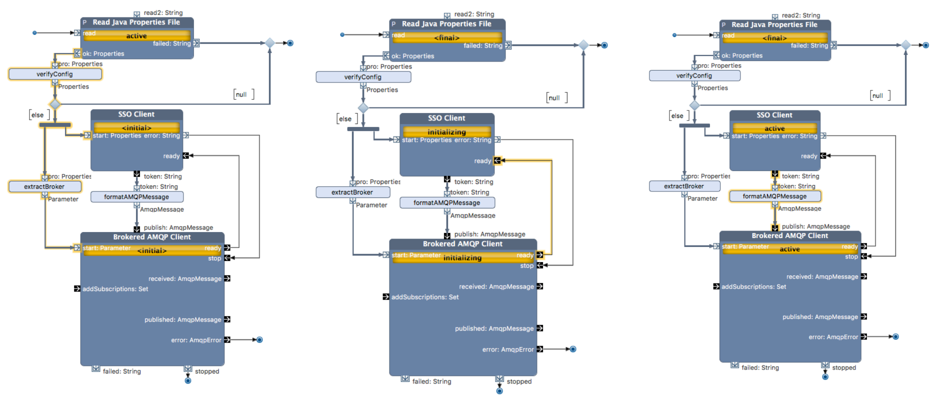


Figure 2.5. Access control

We give a core building block based component for the distribution of access tokens for the solution as shown in Figure 2.5. In this component, we reused two blocks *Read Java Properties File* and *Brokered AMQP Client* to create a new block *SSO Client*. The *SSO Client* block can retrieve a JWT access token from an SSO authentication server (here we use the Telia Identity service [3]), and distribute the token via the AMQP protocol. In the service initiation, the *Read Java Proeprties File* block is started. Here, properties like AMQP host address, port, queue name, Oauth service configurations like the base URL, authentication URL, redirection URL, token URL, client ID and client secret are retrieved from a configuration file. The configuration properties are verified in the *verifyConfig* function. If the verification is successful, the *SSO Client* and the *Brokered AMQP Client* blocks are initiated by forked tokens

[3]`https://developer.telia.io/`



Read configuration properties ready.

AMQP initialization ready and notify SSO Client.

JWT token distribution via AMQP protocol.

Figure 2.6. Animation of main executions for the Token Distribution Component

arriving at each other's *start* pins. If the AMQP block finishes its initialisation, it will emit a token via the *ready* parameter out pin to notify the *SSO Client* block via the *ready* parameter in pin. The *SSO Client* implemented the client side of Oauth service of Telia IoT's identity and consent services, where users can apply their telephone number as their identity and get an access token for the access control. When the system is initialised and ready, the *SSO Client* block can constantly receive or update the JWT access token, and forward it to the *token* parameter out pin. After formatting the token into an AMQP message, the token is sent out to any subscribers of the AMQP message queue via *publish* parameter in pin. The application of AMQP blocks in a IoT solution is universal since it can contain any application messages and data, and the communication is secured by the token contained in the AMQP message since the client can use the public RSA key to examine the validation of the token. Figure 2.6 shows the main execution steps of the token distribution service.

3 BACKGROUND

In this chapter, we examine the use of model-based system analysis and synthesis applied to software engineering. These methods include model transformation, real-time and probabilistic modeling and verification, and formal trends for cyber-physical system analysis, including the spatiotemporal pattern of typical cyber-physical systems, such as remote control systems or automation systems. At the end, we briefly mention the combination of simulation techniques for our verification of real-time probabilistic systems as a compliment for system safety properties.

1 Model and Model Transformation Techniques

The motivation behind model-driven software development is to move the focus of work from programming to solution modeling (see [171]). The Unified Modeling Language (UML, [153]) is the product of many years of standardizing visual modeling notations. UML unifies scores of notations that were proposed earlier and gained significant industry support and became an Object Management Group (OMG) standard in 1997. Model transformation is one of the core activities of model-driven software development. Its main effects include providing a handling mechanism for describing important aspects of a solution with templates to increase productivity and quality. In our practice, we involved model transformation in every part of our research work.

The following paragraphs introduce some typical model transformation techniques that we use in our work.

Attributed Graph Grammar (AGG)

The *Attributed Graph Grammar* (AGG) [189] approach has been developed starting from 1997 and supports the model transformation approach based on attributed graph transformations.

- Supporting language: Java.
- Latest Version: 2.0.6.
- Download link: `http://user.cs.tu-berlin.de/~gragra/agg/`.
- Developer: Technical University of Berlin.

AGG based transformation can support pre-pattern to post-pattern transformation in any Java based object or facilities. A transformation example of a rule-based graph transformation in AGG can be seen in Figure 3.1. In Figure 3.1, The left hand side graph is a condition pattern denotes the cases where this rule can be applied. Here it describes a ownership relation that a pin type vertex belongs to a role type vertex. In the middle of Figure 3.1, the pre-graph describes the initial state before transforming the graph, describing the ownership relation between *collaboration* vertex and *pin*/*role* vertices in a flow-global choreography model (a choreography model is used for describing a distributed systems where multiple roles and multiple components participant in a service, see Paper A). The flow-global choreography model omits the details of flows involved in collaborative services, while the localizing

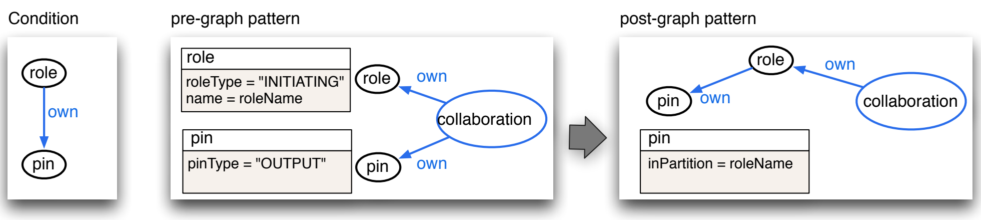


Figure 3.1. Rule-based graph transformation for model transformation from Paper A

policies need to put the flows into certain roles of a collaborative service. The post-graph pattern assigns the ownership of a *pin* vertex to a *role* vertex if the property *roleType* of the *role* vertex is ”INITIATING”, and the post-graph pattern also assigns the *inPartition* property to variable *roleName*. More details the localization policies of a choreography graph can be found in Paper *A*.

AGG has been used in several successful projects that are researching applications of graph grammar rules in UML meta-model transformation [92] and agent oriented software engineering [36, 58].

Henshin

Henshin [14] is an Eclipse project that provides in-depth visual modeling and execution of rule-based EMF (Eclipse Modeling Framework) transformations. Similar to AGG, Henshin considers directed attributed graphs transformation with declarative transformation rules and procedural transformation units. The application of a transformation rule considers recognition of vertices and edges and also conditions and calculations on attributed values. Meta-types of vertices and edges can include all UML metaclasses. In contract to the AGG transformation rule, Henshin uses a single graph to depict and edit a rule. A simple transformation rule in Henshin can be found in Figure 3.2 [61], The application conditions and creating of new vertices are actions with different *stereotypes*. The stereotypes for a transformation rule include *preserve*, *delete*, *create*, *require*, and *forbid*. Applying a rule consists of finding a match of the rule in the host graph and performing the operations indicated by the stereotypes. The *require* stereotype acts as a Positive Application Condition (PACs) and *forbid* stereotype acts as a Negative Application Condition (NACs), while other stereotypes act as actions such as delete, create, preserve.

- Supporting language: Java.
- Latest Version: 1.0.0 (released 16-08-2014).
- Download link:
  `https://github.com/de-tu-berlin-tfs/Henshin-Editor/`.
- Developer: Technique University of Berlin.

ATL Transformation Language

The ATL transformation language [39] is a model transformation language and toolkit developed and maintained by OBEO and AtlanMod. It is the INRIA submitted implementation of model to model transformation (M2M) for the OMG standard of Model-driven Engineering (MDE).

- Supporting language: Specialized transformation language for cross platform model transformation.
- Latest Version: 3.2.0.
- Download link: `http://www.eclipse.org/atl`.
- Developer: OBEO, INRIA.

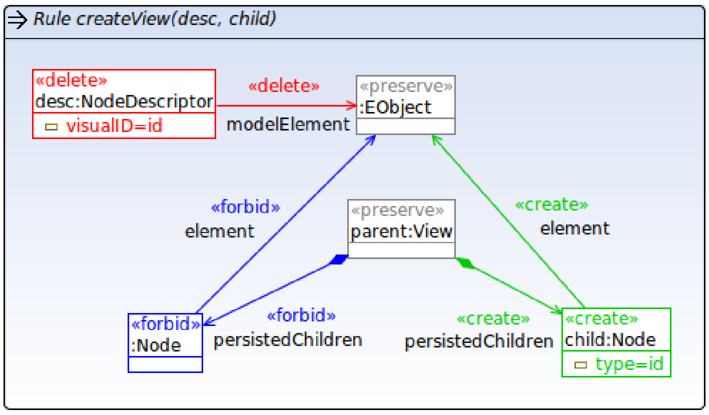


Figure 3.2. Henshin transformation rule [61]

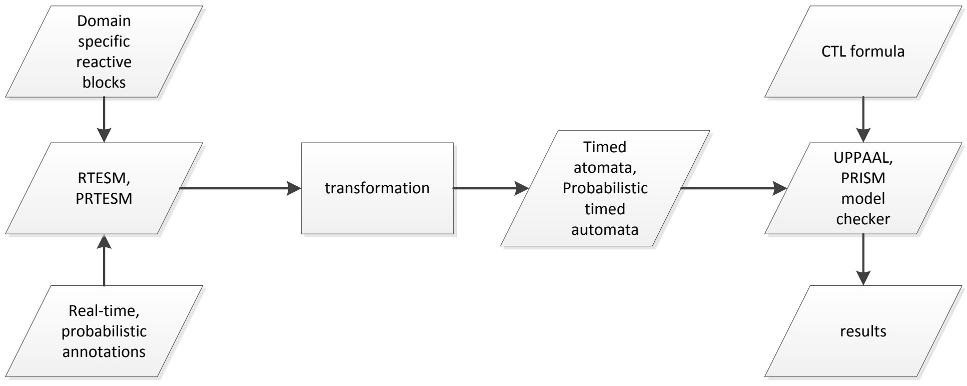

Domain specific reactive blocks
RTESM, PRTESM
Real-time, probabilistic annotations
transformation
Timed atomata, Probabilistic timed automata
CTL formula
UPPAAL, PRISM model checker
results

Figure 3.3. Model transformation-based analysis process

AGG and Henshin are two typical visual model transformation tools that are based on graph transformation and category theory in mathematics.

In our work presented in this thesis, we take a rule-based model transformation approach that applies graph grammar rules to model transformation for component derivation and automatically deduces pattern recognition and remedies. To meet our need for enabling the modeling capabilities we imported the AGG java API for model transformation which makes our Reactive Blocks model transformation customizable.

We also built customized tools as eclipse plug-ins that can be integrated into our model-driven development platform, to translate Reactive Blocks models and additional labels and notations into timed automata or probabilistic timed automata that are recognized by verification tools like UPPAAL and PRISM. As shown in Figure 3.3, the translation includes several steps as follows:

- Sample the meta-elements of the target languages (timed automata or probabilistic timed automata) in the language supported by the corresponding tools.
- Make a one-to-one mapping or composite mapping of the source language to the target language.
- Build the Eclipse plugins for the transformation.

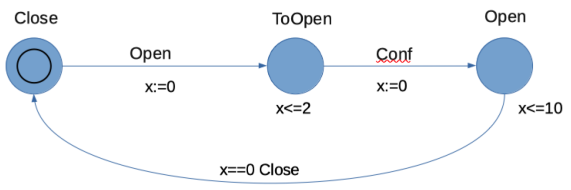


Figure 3.4. Timed automaton [196]

2 Timed Automata and Probabilistic Timed Automata

We extended our modeling and verification capabilities to real-time and probabilistic systems by extending Reactive Blocks with two new formalisms, i.e., Real-Time External State Machines (RTESM) which are based on the timed automata [20] and Probabilistic RTESM (PRTESM) which is based on probabilistic timed automata [104, 121]. By such a translation, the semantics of our building blocks is extended to timed automata (TA) and probabilistic timed automata (PTA), which form the foundation for real-time verification based on Timed Computation Tree Logic (TCTL, [130]) and probabilistic real-time verification based on Probabilistic Computation Tree Logic (PTCTL, [121]). Here we briefly introduce the two formalisms of TA and PTA.

Timed Automata

Following the purpose of modeling real-time properties of reactive systems, we considered a set of formalisms for real-time and finally chose timed automata. A timed automaton [20] is a finite state Büchi automaton [190] extended with a set of real-valued variables modeling clocks. We assume real-valued variables x, y of a set $\mathbb{C}$ standing for clocks and a, b of a finite alphabet Σ standing for actions.

A clock constraint is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \in \mathbb{C}$, $\sim \in \{<, \leq, >, \geq\}$ and $n \in \mathbb{N}$. Clock constraints can be used as guards of transitions for timed automata.

Definition A Timed Automaton (TA) *is a tuple* $< N, l_0, E, I >$ where

- N *is a finite set of locations.*
- $l_0 \in N$ *is the initial location.*
- $E \subseteq N \times \Sigma \times 2^{\mathbb{C}} \times N$ is the set of edges.
- $I : N \rightarrow \mathbb{B}(\mathbb{C})$ assigns invariants to locations.

In the definition, $\mathbb{B}(\mathbb{C})$ denotes the set of clock constraints.

Figure 3.4 shows a simple timed automaton modeling a rail-car door [196]. In the automaton, *Close*, *ToOpen*, *Open* form the location set (N). *Open* (car door open event), *Conf* (event notifying the environment, e.g., a car handler about the door opening, such that some configurations of the service can be removed), and *Close* (car door close event) form the event set E representing the transitions (edges) between states with respective actions (e.g., resetting variable x). The railcar door control system starts at a state called *Close* (l_0), indicated by the double-lined circle. The transition is fired once event *open* occurs, then the system goes to state *ToOpen*, and clock *x* is reset. Within 2 time units (constrained by the state invariant), the system goes to state *Open*. Event *conf* takes place along with the transition. The system remains at state *Open* for at most 10 time units before goes to state *Close*.

Timed automata are the well accepted modelling and verification artifact for time-related properties of reactive concurrent systems. The timed automata model can be easily verified against many properties, e.g., reachability conditions [3], temporal logic [135], and so on. In our research, we applied the most common as well as most important properties like the reachability, that is, whether a system in

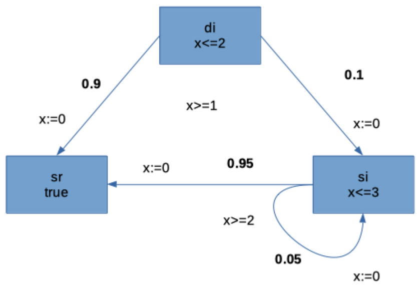


Figure 3.5. Probabilistic Timed automata that models a probabilistic Protocol [123]

timed automata eventually reaches a certain state. The temporal logic properties are expressed in Timed Computation Tree Logic (TCTL, [6]), which is an extension of computational tree logic with time.

Probabilistic Timed automata

Probabilistic Timed automata (PTA) [121] are an extension of TA with probabilities. The formal syntax of a probabilistic timed automaton is defined as follows: Let X be a finite set of real-valued variables called clocks, the values of which increase at the same rate as real-time. The set $CC(X)$ of clock constraints over X is defined as the set of conjunctions over the atomic formula of the form $x \sim c$, where $x, y \in X$, $\sim \in \{ <, \leq, >, \geq \}$, and $c \in \mathbb{N}$ ($\mathbb{N}$ is the set of natural numbers).

Definition A *probabilistic timed Automaton* (PTA) P= $(L, \bar{l}, X, inv, prob)$ is a tuple consisting of the following components:

- A finite set L of *locations* with the initial location $\bar{l} \in L$.
- A finite set X of clocks.
- A function $inv : L \rightarrow CC(X)$ associating an *invariant condition* with each location.
- A finite set $prob \subseteq L \times CC(X) \times Dist(2^X \times L)$ which is a probabilistic edge relations. Or can be called *probabilistic edges* which yields the probability of moving from l to l', and reset specified clocks.

Here, $Dist(Q)$ is the set of functions $\mu : Q \rightarrow [0, 1]$, such that support($\mu$) is a countable set and μ restricted to support(μ) is a (discrete) probability distribution.

An example of a Probabilistic Timed Automata is shown in Figure 3.5 [123]. In this simple probabilistic protocol, vertex *di* (belongs to the location set (di, sr, si) denotes a state in which sender has data, receiver is idling; *si* denotes the state that the sender sent data, however, did not reach the receiver; vertex *sr* denotes the state that the sender has sent data which was correctly received by the receiver. The automaton starts at vertex *di* and after between 1 and 2 time units, the protocol makes a transition either to *sr* with probability 0.9 (data received), or to *si* with probability 0.1 (data lost). In *si* after 2 to 3 time units, the protocol will attempt to resend the data, which again can be lost with probability 0.05. Transitions $di \xrightarrow{0.9, x:=0} sr$, $di \xrightarrow{0.1, x:=0} si$, are examples of probabilistic edges ($prob$) with clock reset and probability. In this way, one can, for instance, compute that data will reach the receiver after at most three time missing with a probability of 99.975%.

3 Modeling and Verification Architecture

The formal verification of models is an important aspect of the model driven development (MDD) paradigm. Many verification methods and tools are designed based on model checking, but there is still a gap between model driven architecture and model verification methods. The main reason is that modeling formalisms need to be translated into formal analysis formalisms for domain specific needs, such that modeling features can be supported by domain specific analyzing tools. However, the modeling infrastructure needs to be extended to support domain specific modeling features, and this requires an extensible modeling infrastructure.

The extensibility of the SPACE approach has been exemplified by previous work and our work in this thesis. Remember, in Chapter *2*, we explained the role of the External State Machine (ESM) which is the formalism for constrain the behavior of the building block with its external environment. The ESM summarises the functional behaviors of the building block and various extension are provided for non-functional behaviors. The modeling and analysis of software reliability of Reactive Blocks are done through Extended External State Machines (EESM) [180] and External Reliability Contracts (ERCs) [182]. In continuation to this, our work declares a model transformation approach for the extension and scalability of modeling and analysis capabilities for our modeling architecture. We proposed and demonstrated a modeling annotation and model transformation framework for domain specific model analysis and verification. This approach is illustrated in Figure 3.3. We defined a performance profile for Reactive Blocks in Paper *D* in which a rich set of concepts for performance modeling is described. The set of concepts includes scheduling mechanisms, modeling of real-time/probabilistic properties by clock variables and guide conditions. By the profiling mechanism, we extend the ESM of Reactive blocks to a Real-Time ESM (RTESM) (in Paper *C*) and Probabilistic RTESM (PRTESM) (in Paper *E*). By extending the domain specific building blocks in Reactive Blocks, we endow the building blocks with new semantics based on timed automata [20] and probabilistic timed automata [104].

Thus, we can use the UPPAAL model verification tool for the verification of reachability properties and translate models from the extended Reactive Blocks mechanism into the verification language of UPPAAL, which is a subset of TCTL. An example of the verification formula can be seen in Section 5.

Further, the modeling capabilities of real-time Reactive Blocks are extended with probabilities by importing the Probabilistic Timed automata. The PRTESM models software system that exhibits stochastic behaviors such as time delays of a safety-critical control action. The PRTESM and fundamental verification mechanism are supported by model transformation and the PRISM verification tool [121]. And an example of PTCTL model checking formula with PRISM can be found in the *PRISM* of Section 5.

4 Theorem Proving

Before talking about model checking, we will mention briefly the theorem proving techniques since they are related. Theorem proving is, in general, used for solving the general validity of a formula, or a problem of whether a formula *F* holds in a mathematical system:

$$\models F$$

Theorem proving utilises the *proof inference* techniques in some *proof system* for solving the problem. First, the problem itself is transformed into a *sequent*, a working representation for the theorem proving problem. A proof system is a collection of *inference rules* of the form:

$$\frac{P_1 \dots P_n}{C}$$

The meaning of an inference rule is that if all the premises P_i are true, then the conclusion C is guaranteed to hold as well.

An *axiom* is an inference rule that does not have premises. In this case, the conclusion automatically holds. A *proof* of a sequent is a *derivation tree* whose nodes are sequents, the root being the sequent to be proved, and for each sequent in the tree, all of its children are premises of some inference rule in which that sequent is a conclusion. The building of a *derivation tree* (*proof tree*) can either be bottom-up or top-down. In the first case, one starts with the axioms from which the original theorem is inferred; In the second case one starts from the theorem as a current subgoal, and then applies inference rules to the current subgoals in order to generate new goals. If the theorems are provable, then the theorem can be expended into such a derivation tree by the inference rules system, and all the leaf nodes of the tree are known axioms or global inferences.

The combination of model checking and theorem proving is therefore often used as the solution for software engineering and verify properties of domain specific models (see [21]).

5 Model Checking

Model checking is an automatic method for guaranteeing that a model of a system satisfies a formally described property [49]. This verification technique has become one of the core research domains for software engineering and theoretical computer science. It covers every specific field of the software development domain and all of the attributes a software system can exhibit, from fundamental properties that a system needs to guarantee (for example, a system should be deadlock-free) to real-time constraints, adaptive behavior, and probabilistic behavior. In this section, we examine the verification techniques we have studied, and applied to our software engineering method.

TLA and cTLA

The SPACE method has applied the *Temporal Logic of Actions* (TLA) [127], which is a form of *Linear Temporal Logic* (LTL), and its verification language TLA+ as the formal background for reactive component based system verification. TLA has been famous for verifying distributed computing systems, and its author Leslie Lamport was awarded the ACM Turing Award for his advances in reliability and consistency of computing systems. In the motivation, ACM mentions TLA as especially useful for imposing clear, well-defined coherence on seemingly chaotic behavior of distributed computing systems [1].

The SPACE method models software components as state machines and distributed services as the communication of state machines. The state machine and communication fits with the semantics of TLA, which models system behavior as a set of infinitely long state sequences $< s_0, s_1, s_2, ... >$, and starts with initial state s_0. To fit with the compositional and incremental specification style, the SPACE method chose an variation of TLA, which is called compositional TLA (cTLA) [95]. cTLA introduces the notion of processes, which can be compositional and describes systems as a combination of other process instances, with specifying a sub-functionality of the system [116].

Real-time Model Checking and Timed Computation Tree Logic (TCTL)

TCTL is a branching-time temporal logic, which extends the classical untimed branching-time Logic CTL [6] with time constraints on modalities.

In our practice of real-time model checking, e.g., we explored several usages of TCTL state duration analysis, worst case execution time analysis. These two typical scenarios can be checked by the common reachability checking using TCTL formulas [3]. The modeling artifacts will be introduced in Section *4* of Chapter *4*, and details of those verifications and results can be found in Paper F and Paper C.

One typical property for a real-time state machine is to measure the elapsed time when some state is reached, especially when we have defined the time duration of the set of actions that build a path to that state. Figure 3.6 shows a simple timed automaton with states s_0 and s_1 and one clock c. The transition from the initial state s_0 to state s_1 consumes a maximum of 5 time units, and the state s_1 is timeless, meaning the duration that the system stays at this state is zero. The TCTL formula for checking whether s_1 can be reached within 5 time units can be stated as shown in the following formula:

$$A\Box(s_1 \; imply \; c <= 5) \tag{1}$$

We do not support all the grammars that a TCTL can express, but mainly the subset of TCTL grammars that are covered by the supporting tools . In addition, we facilitate some extra features that are discussed in Section 3.6.

[1] `http://amturing.acm.org/`

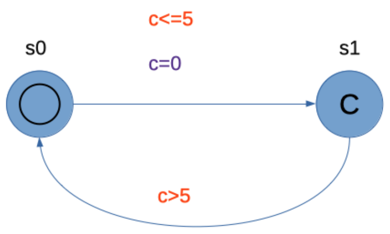


Figure 3.6. A simple timed automaton with one clock

UPPAAL Tool

During the application of real-time extension and verification of real-time properties, we applied the UPPAAL tool [19] for model checking. UPPAAL was developed by the universities in Uppsala and Aalborg for academic use in the verification of real-time systems. It is based on automata with extended modeling capabilities. For instance, it supports data structures of integer variables and C code. Concurrency of real-time systems can be modeled by a network of timed automata with synchronization semaphores. UPPAAL also supports extra features such as committed locations and urgent locations.

- *Committed locations* are labelled with “C” where tokens are not allowed to rest in such locations and no time elapse is allowed.
- *Urgent locations* are labelled with “U” where time may not pass, but can interleave with normal locations. Thus, urgent locations are less strict variants than committed ones.

In our translation, we used a lot of urgent locations to synchronize different automata translated from UML activities and external state machines, the usage of urgent locations are visible in the examples in Chapter *4*.

Further, UPPAAL supports the model verification using a subset of timed computation tree logic (TCTL) [6], and traces counterexamples by simulation. In our research, we use the formalism to extend our Reactive Blocks modeling capabilities and as a tool to aid in verification of properties. For example, in Paper C, we used a case study of an electrical motor controller system to add modeling notations and model transformation from Reactive Blocks to a network of timed automata. Here, we use the UPPAAL supported format to express a subset of TCTL grammars, that is, the verification formulas that are supported by UPPAAL [20]:

Let *TA* be a timed automaton with clock set c and location set *Loc*. For a TCTL-state-formula ϕ, the following operators are used:

- A$\Box\phi$: Invariantly ϕ, which expresses that along all computation (A) and for all points of time ($\Box$), the sub-formula ϕ has to hold. This formula is used to express safety properties.
- E $\diamond$ ϕ: Possibly ϕ, which expresses that some state satisfying ϕ. The path formula E$\diamond\phi$ should be used to check reachable state and is the typical formula for expressing reachability properties in temporal logic.
- A $\diamond$ ϕ: Always eventually ϕ, which expresses that along all computation (A), the sub-formula ϕ eventually hold.
- E$\Box\phi$: Potentially always ϕ, which expresses that for all points of time ($\Box$), there exists a computation (E), the sub-formula ϕ has to hold.
- $\phi \rightarrow \psi$: ϕ always leads to ψ. This is shorthand for $A\Box(\phi \Rightarrow A \diamond \psi)$.

Here ϕ, ψ are local properties that can be checked locally on a state and are boolean expressions over predicates on locations *Loc*, integer variables, and clock constraints c.

UPPAAL supports networks of timed automata that are synchronized by so-called synchronization channels. This feature greatly extended the compositional modelling capabilities and thus reduce the state space for real-time automata models. A synchronization channel with a synchronization initiation automaton and one receiving automaton (binary synchronization channel) or multiple receiving automata (broadcast synchronization channel). A transition in a UPPAAL network of timed automata can have guides (boolean conditions for the transitions to be executed), synchronization (synchronization label that can synchronize different processes), updates (updating variables like a clock) and even more features. A complete list of languare reference for the model can be seen at [193].

Probabilistic Model Checking and PTCTL

Many real-life systems, such as multimedia equipment, communication protocols, networks, and fault-tolerant systems, exhibit probabilistic behavior. This leads to the study of model checking of probabilistic models based on Markov chains or Markov decision processes. Probabilistic Timed Computation Tree Logic (PTCTL) can be used to specify properties of probabilistic timed automata. As an extension of TCTL [6] and PCTL [84], PTCTL can consider the computation of reachability probabilities [123], which is a combination of the algorithms developed for verifying finite-state probabilistic systems [55] and the algorithm for computing the existence of a path satisfying a temporal logic formula in non-probabilistic timed automata [90]. We use the expressiveness of PTCTL to present system reliability properties, and give a few examples of the expressiveness power of PTCTL that are currently supported by the verification tool in the following section.

PRISM Tool

PRISM [120] is the tool for formal modeling and analysis of systems that exhibits random or probabilistic behavior. The input language of PRISM is a formal modular specification language based on the Reactive Modules Formalism [7] which is similar to I/O automata [73]. To provide probabilistic descriptions, PRISM supports discrete probabilities. Each set of state machines extended with clocks and probabilities is modeled in a module and synchronized using synchronization semaphores. It mainly supports symbolic model checking of backward [53] or forward [124] reachability, and the digital clocks approach [122] at present. Probabilistic Timed Automata (PTAs) [102] is the formal formalism with a finite-state automata style, and the discrete probabilistic choice is of Markov decision processes (MDPs) style. We choose PTA as the formalism since it is a natural extension of TA that we use for timed behavior verification in Reactive Blocks.

Besides Figure 3.5, an excerpt of PTAs can be found and is described in Paper E, Section 3.

Calculating the probabilistic reachability: To specify the probabilistic timed logic of PTA, PTCTL employs both quantifiers from Timed Computation Tree Logic (TCTL) [5], which includes a set $\mathcal{Z}$ of *formula clocks*; and quantifier $\mathcal{P}_{\sim\lambda}[.]$ in Probabilistic Computation Tree Logic (PCTL) [84]. Let's explain the quantifiers from TCTL and PCTL and their combinations:

Formula clocks are assigned values by a *formula clock valuation* $\varepsilon \in \mathbb{R}^{\mathcal{Z}}$. Timing constraints can be expressed using such clocks. In the probabilistic quantifier, $\mathcal{P}_{\sim\lambda}[.]$, the brackets can have normal TCTL formulas that express a reachability property, and the quantifier $\sim$ is one of the operators $<, <=, >, >=$ and $=$. λ is a real possibility value and $\lambda \in$ [0,1]. Below we give an example of a PTCTL formula:

$$\mathcal{P}_{<0.01}\ [\Box \text{ error}]$$

This formula can be literally expressed as 'with probability of less than 0.01, an error state is reached'. Or we can check a formula against the PTA model to get a possibility value.

$$\mathcal{P}_{=?}[F_{<=4600} \text{ error}]$$

This formula can calculate the possibility that the *error* state is reached within 4600 time unit. The *F* operator means eventually and also called *future* operator in PTCTL.

In our work, we use the PRISM model checker for probabilistic system behavior verification. In some embedded software system analysis, verification techniques need to consider various software and hardware aspects of the system including communication protocols, digital circuits and controllers. These aspects usually present real-time probabilistic behavior, for instance, a robot may be required to process a task in a predefined amount of time with a probability of 99.999999% to prevent expensive maintenance operations. To address the real-time probabilistic behavior verification so that we can provide safety-critical assumption analysis, we provide a translation of Reactive Blocks with annotations to PTAs, which is the input language for the PRISM model checker. We applied PTCTL to verify against questions, such as can an action of a software controller module be performed within a certain time limit with 99.999999% possibility? We also perform quantitative analyses and answer questions like: What is the probability of a controller fulfilling a time limit in a process?

More details of the case-study and probabilistic system behavior analysis can be found in Chapter *6* and Paper E.

6 Trends in Formal Analysis of Cyber-physical Systems

Model checking for cyber-physical systems and their properties is a complex task since cyber-physical systems comprise software, hardware and environmental factors and compound heterogeneous behaviors. In [30], Blech and Schmidt describe a framework for checking the spatial-temporal behaviors of autonomous cyber-physical systems, such as large distributed remote control systems, including robotics and manufacturing facilities. One approach is to model the systems with distinct synchronization points between different components, where the component is the physically distributed element. The modeling approach includes the following:

- First, a topological or geospatial coordination system is built to interpret or govern the cyber-physical system's overall working geographical environment.
- Second, the static components and their interconnections are modeled.
- Third, the mobile components are modeled and their moving routines are planned in the form of grid-like units.

Using the above modeling infrastructure, the behavioral properties of a system can be analyzed and patterned in relation to the control software of such systems. A spatiotemporal algebra is designed for such analysis purpose [26]. With our real-time and probabilistic modeling and verification capabilities for reactive control software components, we are able to combine the spatiotemporal algebra for the behavioral pattern and safety property analysis of cyber-physical systems [31].

In Paper *G* we delineate a wireless control system for an automatic transportation robot control system. We are able to analyze the spatial safety controls of such system, such as the system's ability to avoid collisions. The analysis contains modeling and analysis of real-time reactive systems, simulation of communication delays in an IEEE 802.11 WLAN environment, and spatial behaviors of remote control transportation robotics. The spatial constraints are verified by the BeSpaceD tool [29], and the simulations are performed by the Jemula802 tool [4].

4

MODELING AND ANALYSIS FOR REACTIVE REAL-TIME SYSTEMS

In this chapter we summarize the modeling and analysis for reactive real-time systems based on the Reactive Blocks paradigm. In Section 1, we present the preliminaries for modeling time and response constrained systems and the novel modeling and analysis requirements in the Reactive Blocks approach. Since we introduced the timed automata and CTL as a pre-condition in Section *3.4* and *3.5*, here we only describe the real-time requirement for the Reactive Blocks formalism. Section 2 summarises how the new approach is synthesized with existing functional modelling with Reactive Blocks. In Section 3, we define the translation rules for transforming Reactive Blocks into timed automata, and provide basic translation patterns. In Section 4, we present a case study carried out for transforming Reactive Blocks into timed automata and real-time property verification, and particularly address when a property needs to be modelled by two collaborative blocks working in a system. In Section 5, we will have a glance on simulation based response speed analysis of reactive systems. The simulation results support the formal modeling of real-time constraints.

1 Preliminary

In our work, we need to understand three preliminaries: Reactive Blocks, real-time requirement modeling and real-time verification. In this section, we elaborate on those aspects.

Reactive Blocks

As mentioned before, the Reactive Blocks approach uses UML 2.x activity diagrams for the modeling of behavior of building blocks. Such behaviors are encapsulated by Extended Communicating Finite State Machines (ECFSM) [141, 142], a kind of finite state machines [116]. Reactive Blocks allows the modelling of distributed communication by token exchange. Since activities have semantic similarities to a Petri net, the translated behavior can be understood as tokens passing places and transitions [113]. When tokens pass places, vertices, and nodes, the corresponding actions, such as Java method calls, are executed. Each transition is strictly triggered by certain events, such as the expiration of a timer or the reception of a signal. The Reactive Blocks approach aims at using the external state machine (ESM) as a strict contract between the software module and its execution environment. The contract is the paradigm for component-based software development as we ideally want to wisely decouple the system behaviors and build complex systems step by step. As the behavioral aspect of systems can be modelled, integrated, and tested against given input and output, a generalisation of such mechanism is to extend modelling, integration and verification to time related systems.

Real-time Requirement Modeling

The modeling and validation of timing constraints for real-time system is introduced in [6] and [1]. In [52], Dasarathy categorizes real-time constraints into two types: behavioral and performance. The first type is a reaction time constraint imposed on system actions or users, and the second type is a constraint imposed on the response speed of systems. Both types of constraints can be modeled using formal language with combinations of timer variables. We also address the modeling of real-time constraints by extending reactive system formalism with timer variables, and giving assumptions and hypotheses on network, traffic protocols, and circuit performance. Dasarathy describes that real-time constraints in a real-time system can be specified with finite-state machines (FSMs), in which a response is always completely determined by the system's present state and the stimulus that has arrived [52].

Since the essence of Reactive Blocks follows an extended finite state machine formalism, the infrastructure provides us with basic elements for modeling time-related features: i.e., the states, transitions, and token resting positions. There are basically three types of constraints we need to capture for time related behavior:

- Maximum time: No more than *t* time units may elapse between two events.
- Minimum time: No less than *t* time units may elapse between two events.
- Durational time: In case of a hard real-time system, an event must occur for *t* time units; for a soft real-time system, durational time bounds of an event can be settled by maximum time and minimum time.

Providing the above basic ingredients and requirements, we need to extend building blocks with time variables and transition guards following the timed-automata semantics, also since we are following the model transformation approach, it is better to provide translation rules and patterns for the target requirement. Since with real-time modeling, system behaviors need to meet not only the functional correctness requirements, but also the time constraints (see Section *3.5*). The challenges exist in how we facilitate an existing compositional specification language to adopt new features while keeping the compositional correctness and guaranteeing the new compositional features.

Real-time Verification

Due to the complexity and computational richness of real-time verification, there are dazzling techniques for verifying real-time properties and many important problems are undecidable [76]. In addition to Timed Computation Tree Logic (TCTL [130]), probabilistic timed computation tree logic (PTCTL [121]) in our work, there are also MTL [160], and RTGIL [144]. Recent trends tend to add observation patterns to real-time models and real-time properties such as reachability can be verified but with a minimised state space [74].

In our work, we extend the external contract with time requirements expressed as clock variables with constraints, guides, and invariant condition on states according to the timed automata paradigm; The internal implementation are labelled with clocks constraints that expresses the protocol assumptions or operating system supported real-time features, e.g., the worst case execution time (WCET). After extending the external contract and label the internal behavior, we need to check if the internal implementation fulfil the contract, then the component is contract fulfilled component, and we say the behavior of the component can meet the required real-time need. This requires formal verification to be applied to the component. Further, we think carefully about the real-time property propagation in the compositional manner, i.e., can a real-time property be properly modelled and abstracted in an external contract? Moreover we considered the translation of multiple components to timed automata when verifying real-time properties.

We answered the above questions in Section 2 and Section 3. In Section 4, we exemplify a complementary translation when the properties needs to consider real-time properties that are expressed with two collaborative building blocks together. Alltogether, we mainly carried out worst case execution time (WCET) analysis, probability analysis for real-time verification and real-time probabilistic verification of Reactive Blocks.

2 Extended External State Machines with Time and Probability

External state machines (ESM) have served their purpose in Reactive Blocks and we found it extremely extensible. Existing extensions for the ESM include [179] in which Slåtten introduced the Ex-

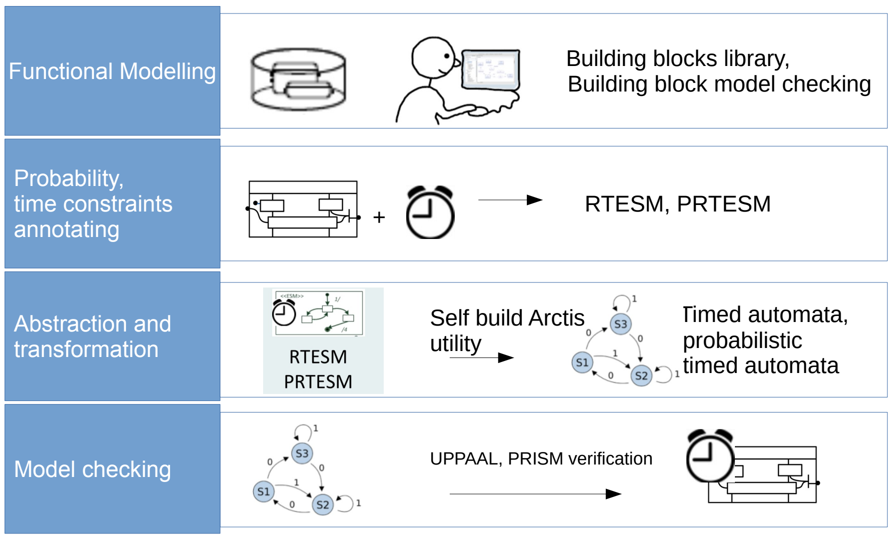


Figure 4.1. Approach for creating time-constraints correctness preserved block

tended ESM (EESM) and External Reliability Contract (ERC) for specifying reliability properties of Reactive Blocks.

To achieve real-time and probabilistic property modeling and verification, we also extended ESMs, but with real-time and probabilistic annotations. The extensions are developed by a series of activities that involve modeling, annotation, transformation and verification. We describe our approach for creating correctness-preserved real-time reactive system components as delineated in Figure 4.1:

- **Functional modeling**: First and foremost, we use Reactive Blocks to model the behavior of embedded real-time systems that are typical in robotics and embedded automation systems. Here, the control state can be usually modeled using ESMs, and detailed behaviors can be implemented within a building block. Software engineers who are familiar with software development and embedded control systems are qualified to perform this step.

- **Probability, time constraints annotation**: Next, we analyze the real-time constrained behavior and extend the ESM to an RTESM with clock variable evaluation statements, guard notations, and state invariants. This can be done, for example, by engineers with knowledge of real-time system modeling and embedded device real-time standards. Of course, a 100% guarantee that a hard real-time threshold is always kept, can never be given since the control system itself can fail with a certain likelihood. To handle probabilities for real-time constraints, we provide the PRTESMs in which the engineers can annotate the real-time constraints with probabilities.

- **Abstraction and transformation**: With our self-built tools for connecting our modeling framework to verifiers of different kinds, the time-constrained control behavior described in the RTESMs and PRTESMs are translated into input to UPPAAL and PRISM for timed and probabilistic behavior verification. Properties such as the constraints of a series of durational actions can be checked by the UPPAAL tool; For probabilistic behaviors, we seek to verify the maximum (or minimum) probability that an unsafe action may exceed its designated maximum time. The probabilistic properties can be described by Probabilistic Timed CTL (PTCTL) (see Section *3.5*).

- **Model checking**: We analysed the fulfillment of the Reative Blocks against the extended contracts by asking whether the abstracted behavior described in RTESMs and PRTESMs are fulfilled by the implementation; Further the contract has to be a correct abstraction of the real-time properties.

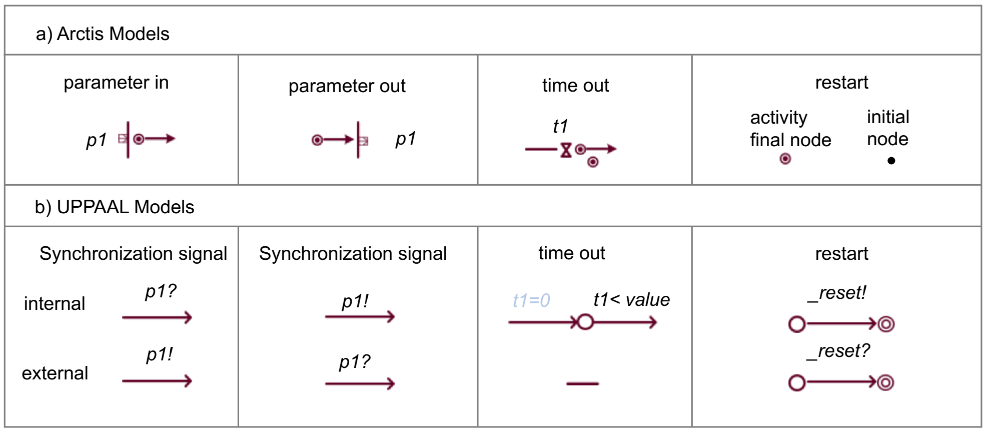


Figure 4.2. Mapping between the Activity Behavior and TAs

Through the analysis process we proved the correctness of the abstraction of real-time behavior, and we provide a way for describing real-time and probabilistic aspect of system behaviors.

3 Translation Procedures

The translation between the extended Reactive Blocks formalism and the real-time verification formalism is the critical link of the chain for the real-time extension of Reactive Blocks. In order to endow the additional annotations with a real-time relevant semantics, we carefully designed the translation. For model checking purpose, we first consider formalizing Reactive Blocks with time variables and constraints into network of timed automata to check the fulfilment of the internal behavior to its RTESM; Second, we considered translating a more complex case in which more than one reference block had to be analyzed for their real-time properties.

Mapping internal behavior, ESM to Timed automata

In the mapping between the building blocks and the network of timed automata in the UPPAAL tool, the ESM and the activities are mapped to two automata, that we call internal TA (timed automaton) and external TA. The two automata are synchronized using synchronization channel. The mappings are shown in Figure 4.2:

- In general, a building blocks behavior is represented as an *internal* TA, while its ESM are represented as an *external* TA.
- A token arriving at an activity via pin *p1* is translated to a binary synchronization channel in which the external TA executes a send signal (*p1!*) modelling the input from the environment while the internal TA carries out a receive signal (*p1?*).
- A token leaving an activity via pin *p1* is mapped to a binary synchronization channel in which the internal TA executes a send signal (*p1!*), and the external TA a receive signal (*p1?*).
- A timer *t1* in an activity is transferred to a node of the internal TA, which is provided with an environment clock variable that is set to 0 by the upstream edge to this node. Further, a clock

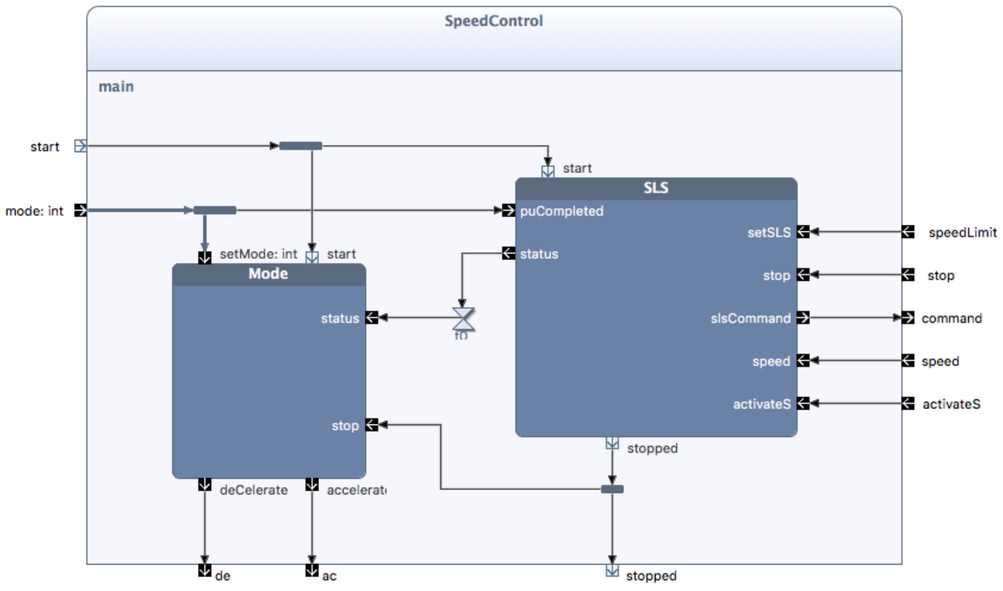


Figure 4.3. UML activity of building block *SpeedController*

invariant states that the TA may be in the state only if the upper bound limit of the clock is not yet reached (expressed as $t1 < duration$).

- A terminated building block, e.g., by a token reaching an Activity Final Node (◉), may be restarted at any time. To model this property in the TAs, we add special reset transitions from the final node to the initial node using a broadcast channel $\_reset$.

The most important part of the translation is that we flattened the compositional specification of Reactive Blocks into a networked timed automata that can be formally verified in UPPAAL as a network of Timed Automata. The flattening includes the internal behavior of the block (*internal* TA) and ESM (*external* TA). The synchronization of the two TAs is done with sending and receiving signals which are the synchronization channels in UPPAAL. Notice, that in some scenarios, we need to translate more than just *internal* and *external* TAs, e.g., when analysing a reference block, i.e., call behaviors from an third block, we need to synchronize the ESM of the reference block and its containing block's internal behavior and external visible interface (ESM). In such cases we need to import the so called broadcast synchronization channel to synchronise more than two TAs.

4 Translation Example

We carried out several case studies in automation control and robotic control systems, e.g., the safety motor controller block in Paper C and the *ControlTwoElements* block in a transportation robot in Paper F. In the following discussion, we introduce the translation through a case study of *Speed Controller*. In this example, we give a transformation in which we consider the transformation of two reference blocks with their behaviors synchronized with a container block.

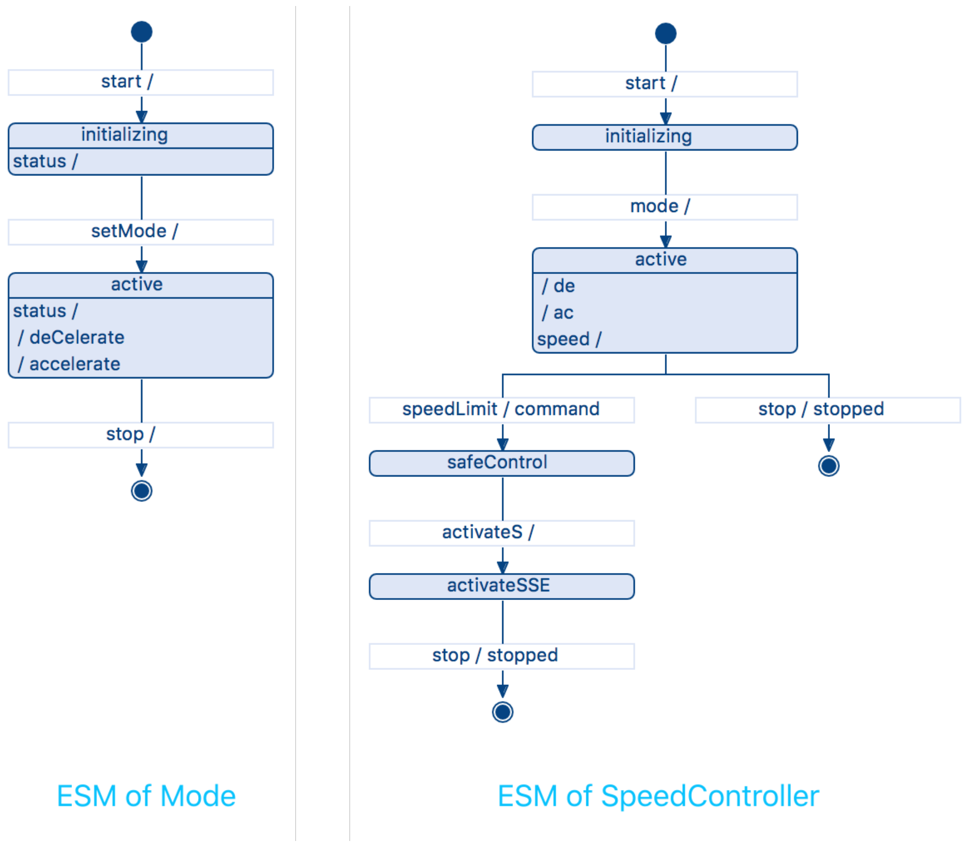


Figure 4.4. ESMs of the block *Mode* and *SpeedController*

Component Modeling via UML Activities

Figure 4.3 shows a building block for a simple speed controller[1]. The UML activity diagram shows the behavior of the building block *SpeedController* that we use to specify the speed controller. This block contains two reference blocks *Mode* and *SLS*. The activity of *speedcontroller* is started via the token arriving at the pin *start* initiating the two reference blocks. Thereafter it waits until a token arrives at the pin *mode* which effectively activates the two reference blocks. When active, the block *SpeedController* can receive tokens from *speedLimit* (set the speed limit), *speed* (receive speed report) and *activateS* (activate the safe emergent stop) pins, and emit tokens out via *de*, *ac* (decelerating resp. accelerating speed) and *command* (send command to the environment) pins; finally the block is stopped via a token arriving at the *stop* pin after which a token is sent to the environment via *stopped* pin. Note, that there is a timer t_0 in *SpeedController* which separates the actions between the two reference blocks. The *Mode* block specifies three different controlling modes of an electronic motor, which are *normal*, *sports* and *economic* [2]. The mode is set via a integer value and default to be *normal*. The external behavior of the *Mode* block is simple and depicted by its ESM in Figure 4.4. The *Mode* block is started by a token passing pin *start* and reaches the state *initiating*. After the block is fed with a token from the *setMode* input pin, the block reaches the *active* state in which it can constantly emit tokens out via *deCelerate* and

[1]The Simple speed Controller was originally developed by Hien and Braek for an European traffic controller system [108].
[2]This building block is inspired by the state refinement example in [27]

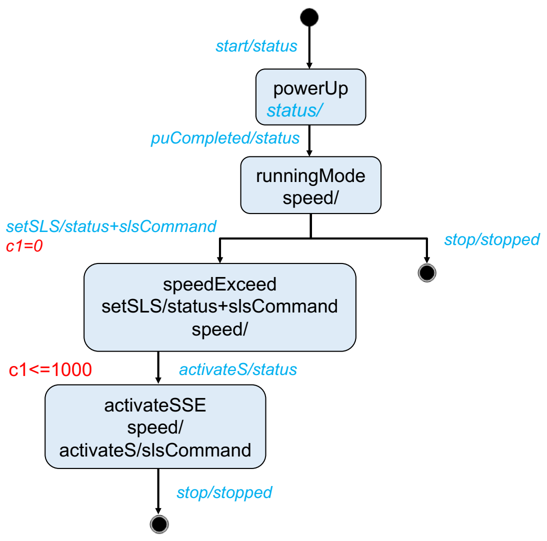


Figure 4.5. Simplified (RT)ESM of block *SLS*

accelerate. During the life-cycle of building block *Mode*, it can always receive report information from the *SLS* block via the pin *status*.

The block *SLS* is a simplified *Safe Limited Speed* Controller (A more complete version of the *SLS* can be found in Paper *C*). We extend the ESM of *SLS* with clock variable *c1* and annotate the ESM with clock reset and guide notations so as to format the RTESM of *SLS*. The RTESM of *SLS* block is shown in Figure 4.5. This network of timed automata shows the composition of the two sub-blocks and their synchronization with the system. The four states are of the RTESM are explained below:

- *powerUp* The motor control system is starting.
- *runningMode* The motor is running normally and below limited speed.
- *speedExceed* The motor runs above its permitted speed limit but did not yet exceed the maximum time period after which it has to be shut down by executing the Safe Stop Emergency (SSE) handler. On the incoming edge towards the state *speedExceed*, the clock variable c_1 is reset to zero (in red color) for counting down the actions to trigger the state *activateSSE*. Assuming the maximum time for the speed exceeding safe speed limit is 1000 time unit. To give such constrains to this action, at the outgoing edge from *speedExceed*, a guide condition $c1 <= 1000$ is labeled.
- *activateSSE* The SSE handler is triggered and the motor is shut down.

The parameter pins of the *SLS* block are listed blow:

- *start* The building block is started via a token passing in the initiating pin *start*. Then the *SLS* block goes to the *powerUp* state.
- *status* The *SLS* can constantly emit tokens out via the *status* pin, in the speed controller system, and the status is reported to the *mode* block.
- *puCompleted* After the speed controller system set the operation mode, a token is fed to *puCompleted* parameter pin to notify *SLS* block that the preparation is ready and *SLS* block goes to the state *runningMode*.
- *setSLS* By tokens passing this pin, the maximum speed limit may be altered.

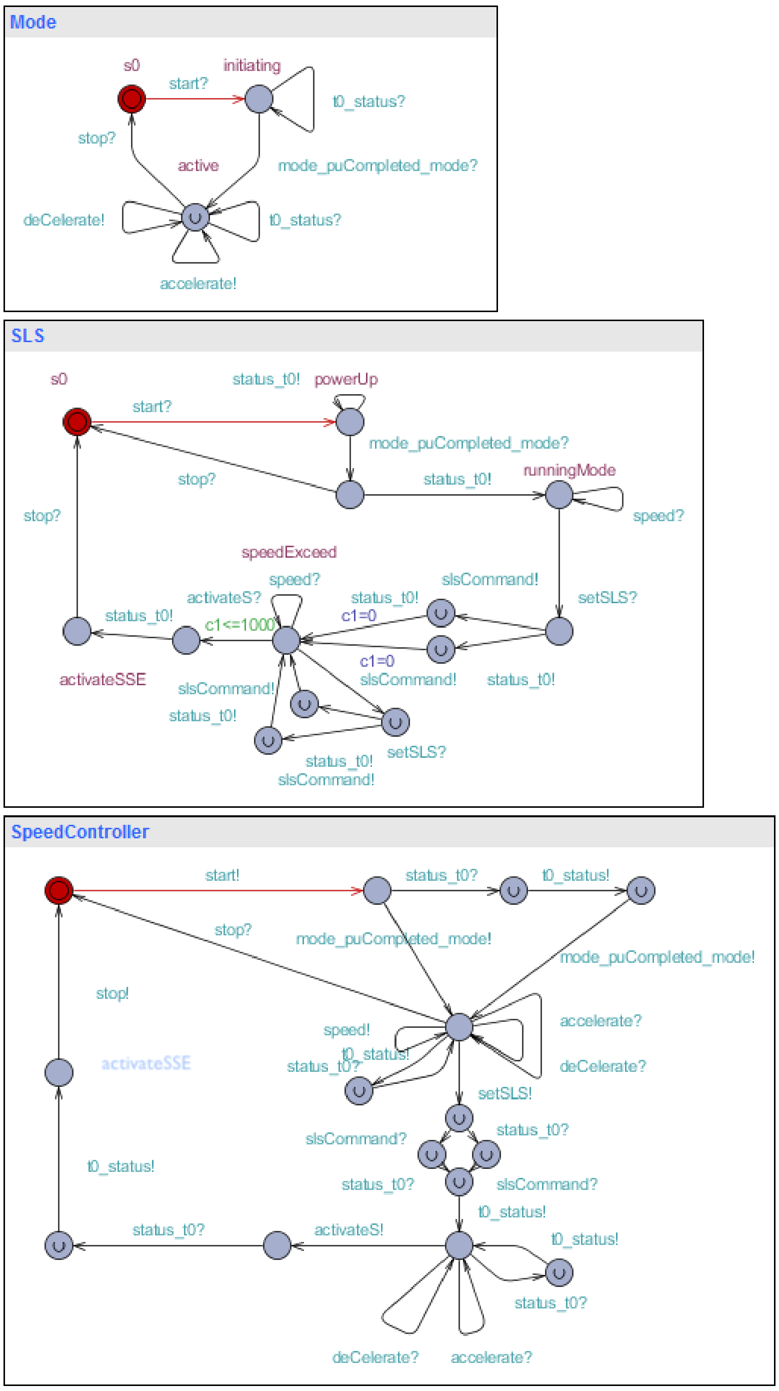


Figure 4.6. Network of timed automate in UPPAAL obtained by translating the *SpeedController* block

- *slsCommand* The transition from state *runningMode* to *speedExceed* specifies that whenever a *setSLS* incoming token is received, a token will be emitted out from *slsCommand* pin.
- *speed* When the *SLS* block is in states *runningMode*, *speedExceed* or *activateSSE*, it can receive a *speed* report from the environment.
- *activateS* A token passing this pin activates the Safe Stop Emergency (SSE) function which can stop the motor.
- *stop* Flows passing this pin switch off the SLS block.
- *stopped* This pin is a notification that the SLS block is terminated. In the *SpeedController* block, the token will be forked with one token to terminate the *mode* block, and the other notify the outside environment.

The overall behavior of *SpeedController* block is depicted in its ESM in Figure 4.4. The block is started via a token arrive at the *start* pin and then receive the *mode* setting by the *mode* parameter pin. It contains states *active*, *safeControl* and *activateSSE*. Figure 4.6 shows the set of TCTL models obtained by translating *SpeedController*, *SLS* and *mode*. We can use UPPAAL to simulate the behavior of the three automata together with the timer variable c_1. In the figure, the automata *SpeedController*, *mode* and *SLS* correspond to blocks *SpeedController*, *mode* and *SLS*. and the three automaton are synchronized via multicast synchronization channels. We can see the automata are similar to the ESMs of the composed blocks albeit there are temporal and urgent locations generated during the translation (for more information about urgent location see Chapter *3* Section *5*).

Model Checking Real-time properties of a Reactive Block

In the *speed controller* example, we demonstrated that the behavior of the system expressed as *SystemTA* still preserve the timed properties expressed by the RTESM of the *SLS* block, i.e., the implementation fulfils the RTESM. This is done by rechecking the property:

A[] SpeedController.activateSSE imply (SLS.c1<=1000)

The *activateSSE* state in the *SpeedController* block corresponds to a state where the same real-time property needs to be preserved in the TA of the global system, and this property can be literally expressed as

When the global block SpeedController is in state activateSSE the timer c_1 in SLS is always within 1000 time units.

or in an equivalent expression, it can be expressed as

the action for reaching the Safe Stop Emergent state from the moment the system enters the speed exceed state is no more than 1000 time units.

We abstract the real-time properties of a building block into an RTESM, which is a real-time contract between a block and its environment. The overall work step includes model annotation, model transformation, model checking for real-time properties in a compositional system:

1 The real-time property is expressed using RTESMs that extend the ESMs of the block with timer variables and guide conditions and updates.

2 We translate the internal behavior of the system block, its ESM and an environment to a network of timed automata such that they form a network of timed automata (TAs). We use the model checker UPPAAL to simulate the real-time behavior of a building block together with its environment.

3 We use the UPPAAL model checker to prove that, for a block S, timed constrained behaviors (expressed as timed properties in Timed-CTL (TCTL)) constrained to S's RTESM are indeed fulfilled by its internal design. I.e., we verified that the automaton expressing the internal behavior of S and the RTESM automaton together with an environment automaton have expected consistency in real-time behaviors.

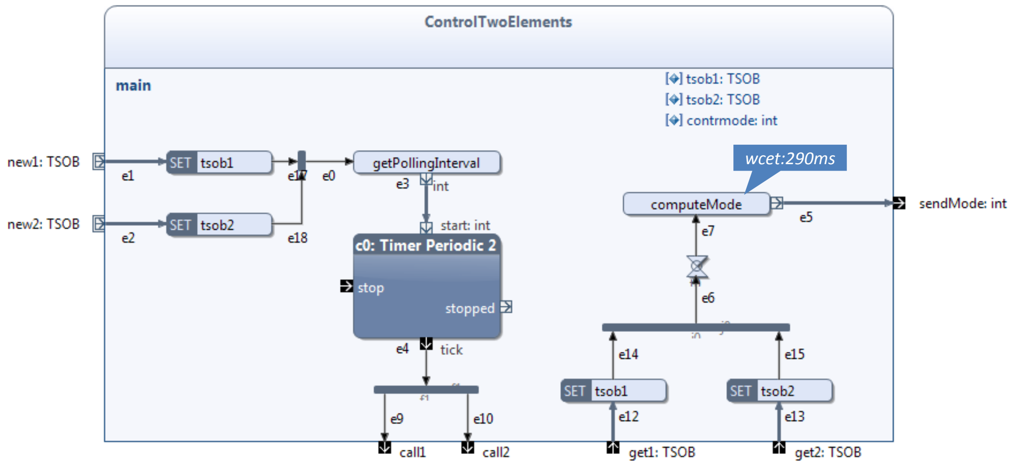


Figure 4.7. UML activity of building block *ControlTwoElements*

Model Checking of Real-time system for Human Machine Collaboration

In the context of real-time systems, when systems are modeled as timed automata, and the state space is infinite due to continuous clock variables, many problems are undecidable. Usually a class of models is defined as decidable whenever checking its reachability properties is decidable [35]. Thus, reachability checking ([3]) is one of the fundamental techniques for model checking of timed-automata. Since the building blocks are translated into networked automata for the verification of real-time behaviors, we can apply reachability checking for safety properties of Reactive Blocks.

In paper F, we carried out another case analysis for a control block for an automatic transportation system in a human machine collaboration environment (so called cyber-physical system). The core function for the control system is a block *ControlTwoElement* (Figure 4.7) which synchronizes inputs from sensors for robot position and working personal distance. It is a feedback controller that polls the sensors of the the robot and the human, and uses the sensor data to compute the correct control mode of the robot. The activity is initiated by two simultaneously arriving data tokens via the parameter nodes *new1* and *new2* containing location information about the robot and the human at system start. The corresponding time, location and speed data is defined by the Java class TSOB which is the type of both parameter nodes. In the same activity step, the two data tokens are stored in the variables *tsob1* and *tsob2* and the two tokens are joined to one passing operation *getPollingInterval*. This operation refers to a Java method that reads out a parameter and pass it to the building block *Timer Periodic 2* that will issue timeouts in periods according to the parameter. The polling is sent out via parameter out *call1* and *call2*.

The sensor data are parameterized in through pin *get1* and *get2*, stored in local variable *tsob1* respective *tsob2*, and joined in a flow breaker. In a new activity step, the token leaves the flow breaker and causes the execution of the Java method *computeMode* which takes the sensor data from the variables *tsob1* and *tsob2* and computes the mode according to the current distance between robot and human. The RTESM and translated timed automaton of the *ControllTwoElement* can be seen in Paper F.

The operation *computeMode* in the activity is annotated with a *wcet* (worst case execution time) of 290 ms since it contains the code to process the execution mode from the sensor inputs. The block has to guarantee that a emergent stop can be carried out if the working personal is too close to the transportation robot. In the building block, we express a typical property using TCTL as follows:

A[](external.computing imply c2 <= 310)

The safety property is expressed in the RTESM and can be literally described as

When the ESM of the building block is in the state computing, can the clock c_2 be always be smaller or equal to 310 time units?

Since the RTESM is a real-time behavior contract of the building block, we have to guarantee that the compositional system is indeed guaranteeing the expressed properties of a real-time behavior no matter whether the real-time behavior is from a reference block that is inside the system building block or it is a system behavior which is implemented by internal activities. This requires that the UPPAAL proofs to carry out two step verification:

1 The UPPAAL proof has to be carried out verifying that the inner behavior of a block is a refinement of the RTESM.

2 The properties $\mathcal{P}$, which are guaranteed by an RTESM, have to hold together with its environment in a composed new system. That is, $\mathcal{P}$ has to be guaranteed by the environment activities together with additional clock variables, events and guide notations.

Both steps of proof can be done by the transformation and the formatted network of timed automata approach introduced in this thesis. And since this transformation are fully automatic and extensible, the timed / probabilistic behavior are also standardized using RTESM / PRTESM. The approach significantly improved the efficiency for compositional development of timed / probabilistic systems. The compositional verification of timed probabilistic properties are similar to real-time properties, while algorithms for compositional verification of probabilistic timed behaviors are designed out in PRISM tool [137].

5 Simulation-based analysis and Verification for Cyber-physical Systems

Simulation of large complex network systems is the main research method for observing and analyzing network based communication systems. In some safety-critical systems, such as Cyber-physical systems, such methods provide an inexpensive means to reveal how the communication protocols are working and how they affect the system performance. Examples of such systems include remotely controlled robots, medical devices, and automated manufacturing facilities. In our safety-critical system analysis, we included simulation-based network protocol analysis, typically analyzing the IEEE 802.11 series of protocols in Paper G.

The *BeSpaceD* framework [29, 30] is a tool for verification and analyzing of special-temporal behavior of component based systems. The *BeSpaceD* tool collects verification results from, for example, the solvers SAT [133] and SMT [57], and allows one to verify spatiotemporal properties like range coverage or the absence of collisions between two components. In Paper *F*, we present a case study for the spatiotemporal behavior analysis of probabilistic timed systems. The *BeSpaceD* tool is integrated into our building bock-based component development environment and the probabilistic behavior of the component based system is analyzed using PRTESM with PRISM as a verification basis. The spatiotemporal behavior is analyzed using *BeSpaceD*.

Probabilistic real-time proves are typically useful in cyber-physical systems where control software interacts with environment via a robot or another physical component. The control system typically presents stochastic behaviors and is heavily affected by the environment. In the case study of the probabilistic real-time behavior of cyber-physical systems in Paper E, we combined simulation results with our software control system model. The simulation provides maximum execution time of a control signal in a safety-critical control environment, from the system side. We applied real-time constraints analysis such that a disastrous failure can be avoided with a next to 100% probability (e.g. 99.999999%). In paper G, we selected the simulation and emulation tool Jemula [4] to simulate a industrial control environment under the popular IEEE 802.11 series of WLAN protocols. We use *BeSpaceD* to analyze spatiotemporal safety properties in our tool chain.

5 RELATED WORK

This thesis relates to model transformation, real-time system modeling and verification, compositional specification for real-time systems based on timed automata, performance evaluation, and complex system engineering. In this Chapter, we first briefly summarize the Reactive Blocks approach in Section 1. Then in the following sections, we take a brief view of the works of related domains: In Section 2 we sketch model based analysis that needs to be assisted by model transformation mainly translating UML based models to various mathematical formalisms, e.g., temporal logic and petri net. In Section 3 we take a look at some basic approaches for rule based model transformation that apply automation to software engineering process. Sections 4 and 5 explore the real-time system modeling and probabilistic system modeling realm. In Section 6 we briefly summarise the performance modeling techniques. In Section 7 we discuss the tentative usage of modeling techniques for cyber-system safety.

1 Synthesis Reactive Blocks for Complex System Engineering

Model-Driven Design has been promoted as the means for coping with the complexity of software systems, it provides mainly two promising approaches for solving the main problems faced by software industry today: raising abstraction level and introducing more automation in the process [143, 185]. Here we summarize some key aspects in the Reactive Blocks approach in raising the abstraction level and introducing more automation in the software development process. The solved key aspects include distribution, compositionality, code generation, evaluation, reliability, and security:

- Distribution: In the context of software distribution, our goal is to build and configure component-based software systems. A way to achieve that, is to start with a high-level specification, a so-called choreography model, that abstract the distribution aspects. The idea of such choreography model (distinguished from the Business Process Execution Language (BPEL) [172] and Business Process Model and Notation (BPMN) [77] choreography model) is originated from Castejon's work on modeling collaborations in service engineering [140] since choreographies are used to model the service composition, i.e., the interaction protocol between several partner services. Kathayat [106] and later Fatima [65] transform requirements into service specifications. A global choreography model is used for high-level software system specifications. Taking the global choreography model as a starting point, we apply rule-based decomposition policies in automatically instantiating the global choreography model to the localized choreography model as a distributed component model ($PaperA$ in the thesis).

- Compositionality: A core concept of MDD for complex systems is its ability to make the definition of compositional models possible. MDD allows systems to be analysed and generated incrementally. In this way, one can decouple a system into a hierarchical component-based system. This aspect is solved by Kraemer in his thesis "Engineering Reactive Systems, A Compositional and Model-Driven Method Based on Collaborative Building Blocks" [110]. His work also contributed to the core Reactive Blocks approach for software engineering.

- Code generation: Code generation is another core activity of MDD. The Reactive Blocks approach solves the code generation problem by automatically translating UML activities into executable state machines from which Java code is fully automatically generated, see [116, 113].

- Evaluation: Originally, the evaluation of a system meant the evaluation of source code, but the term has been extended such that if also refers to the evaluation of models in the MDD paradigm. The evaluation is often a task for software systems due to the complexity of large software-centric systems and various properties to be measured [37]. The fundamental properties of a software artifact include size, length, complexity, cohesion, and coupling, etc. MDD solves the size and complexity issues of software systems by decoupling them into functionally separated building blocks. By providing the components with suitable interfaces, one can then reduce the complex evaluation of the overall system into much easier evaluations of the components. Kraemer et al. have solved the problem of evaluating a compositional building block by defining the External State Machines (ESMs) as behavioral interfaces, and formal reasoning to guarantee that properties described by the individual building blocks are preserved by the composed system [117]. In [177], a tool translating UML activities into TLA+ [201], the syntax of TLA, is presented. This allows us to apply the tool TLC [201] to check various temporal properties that are stated in form of theorems. The evaluation content also changes based on different system requirements. For a large distributed system, performance can be a critical evaluation criteria when reliability, safety and dependability of systems become important issues of a reactive system [191]. For a cyber-physical system, real-time performance, robustness, and safety may all be very important since such a system might be working in a human-machine cooperative environment where a failure might have disastrous consequences. We contribute to the evaluation of safety properties related with real-time performance of a networked system in the Papers *D, E, F,* and *G*.

- Reliability: A big challenge for large software systems is the reliability of critical system components. In his thesis [179], Vidar Slåtten helped the developers to find a way to handle reliability issues by integrating automatic reasoning facilities to discover system pitfalls and design mechanisms to bury those pitfalls. The mechanisms integrate formal reasoning and Reactive Block libraries. In particular, Slåtten introduced the Extended ESM (EESM) and External Reliability Contract (ERC) for specifying the reliability properties of Reactive Blocks in [182].

- Security: Integrated security mechanisms are important for large complex systems. Linda Gunawan extended the Reactive Blocks approach to the domain of security-enhanced embedded networked applications. In particular, a set of security enhanced building blocks are provided to the block libraries, and basic security mechanisms are encapsulated into building block to support security-aware system development, In [79], the application level security properties are expressed also using Extended External State Machines (EESMs, see [182]), embedded system security properties can be analysed using the interface contract instead of analysing system activities which significantly reduced the state-space when applying model checking, other related work can be fouind in [82, 81, 80].

In general, the Reactive Blocks approach provides a method for reusing reliable and high quality software components. This approach can be applied to Complex System Engineering by applying modular development and incremental integration of complex systems.

2 Model Transformation for Model-based Analysis

Model transformation, especially translating and endowing formal semantics to the UML modeling language is a core research activity in MDD and model-based analysis. The UML activity diagram, which describes the global ordering of atomic pieces of behaviors, is popular in system behavior modeling since it can describe complex processes that have parallelism, loops, and event-driven behaviors in a rather comprehensible manner. In [8], Ermeson et. al. translated the UML activity diagram with MARTE (Modeling and Analysis of Real-Time and Embedded systems) profile to a special Petri Net based model ETPN (Time Petri Net with Energy constraints) which can be used to validate functional, timing and low power requirements in early phases of the embedded system development life-cycle. Model checking verifies whether some given finite state machine satisfies some given properties which are specified in temporal logic or other formal techniques. Thus model checking techniques are applied widely to software engineering and model-based architecture. The main problem for applying model checking is the state space explosion for real-life systems. Since the state space of a real-life system is often too large for an efficient verification, a lot of effect is taken to reduce it. A typical technique applied to reduce the state space is to encode it symbolically using predicates [64, 87, 120, 123, **?**].

In [64], Rik Eshuis introduced the translation of UML activities to a symbolic model verifier named NuSMV. In [113], Kraemer and Herrmann present an algorithm to translate UML activities into state machines. In this work, collaborative services are used as the main specification style of interactive behaviors, such that system behaviors are described compositionally.

In [136], López-Grao et. al. translate the UML activities model into a petri net based model for software performance analysis. By translating general UML activities into a Labeled Generalized Stochastic Petri net (LGSPN), they endow UML activities with formal semantics. A similar semantics was added to the UML activities-based software model in our approach, such that we can directly take advantage of such semantics and translate the formalism into other languages that can be analyzed. In their approach, the resulting LGSPN performance model is directly imported and processed in the GreatSPN tool [47] to perform a quantitative analysis and to obtain performance rates. In our approach, we apply a UML profile for performance annotation directly to our building blocks, and the state space of a building block becomes a performance analysis model that can be analyzed by model verification tools such as UPPAAL or PRISM (see Paper C, D E, and F).

3 Rule-based Model Transformation and Verification

Nowadays, the Eclipse Modeling Framework (EMF) is a popular platform for software engineering-based research and model driven development of architectures. We have to thank the authors of [60], who provide a fundamental infrastructure for developers and researchers. EMF is an Eclipse plugin-based project that provides a modeling and code generation framework for Eclipse applications based on a structured data model. In [24], Biermann et al., provide an EMF model transformation framework (EMT) to support the modification of EMF models based on graphical EMF model transformation rules. EMT now consists of three components: a graphical editor for EMF model transformation rules, a compiler, generating Java code from these rules to be used in further projects, and an interpreter for the execution of the rules using AGG [189], a graph transformation tool environment.

The AGG (Attributed Graph Grammar) system provides a formal foundation based on the algebraic approach to graph transformation [50, 62], and offers validation support like graph parsing, consistency checking of graphs and graph transformation systems with conflict detection of graph transformation rules [189]. GROOVE is the project for an integrated graph transformation and model verification environment that supports graph transformation-based model transformation. Its main purpose is to support the modeling of object-oriented systems with a formal foundation for dynamic semantics. GROOVE supports the model transformation of one monolithic graph and explores the state space of a reachable graph. Furthermore, it supports CTL and LTL-based model verification of the state space [105, 75].

New trends for model transformation address voluminous models corresponding to the big data era (e.g., [138]). To meet the need to process large-scale graph data in increasing numbers of application domains, Krause et al. mapped graph transformation rules and units to the Bulk Synchronous Parallel (BSP) model and processed several applications in a distributed and parallelized environment [119]. This can be helpful in many application domains such as model management, model transformation, and software architectures.

4 Real-Time Modeling and Verification

Real-time software systems tend to be large and complex. This time-related complexity exists in a vast spectrum of very different types of systems ranging from purely time-driven to purely event-driven ones. To present the common requirement to respond correctly to inputs within acceptable time intervals, researchers studied systems ranging from telecommunication and aerospace to robotics and many other industry domains since, over time, each of these types of systems has developed its own idioms, design patterns, and modeling styles that collectively capture the distilled experience of many projects. In earlier times, the ROOM method [170] was used with UML to model real-time systems in structural and behavioral aspects. For example, they urged using a UML collaboration diagram to model communicating entities (or actors) and use connectors to model signal-based communication channels that interconnect two or more ports.

In [207], the author proposes an approach that reuses model verification tools and extends the UML modeling notations to real-time (UML-RT) such that the analysis and model were brought closer together and models could be abstracted in a formal representation. We primarily concentrate on real-time systems that are characterized as complex, event-driven, and distributed. The first effect, that needs to be mentioned, is the general-purpose UML modeling language to define and capture this domain-specific usage of real-time reactive systems. The UML community developed the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [152]. MARTE provides a common way of modeling both hardware and software aspects of a real-time embedded system in order to improve communication among developers, and also enables the interoperability between development tools used for specification, design, verification, code generation, etc. We also enabled our approach to real-time analysis by incorporating the MARTE profile from an abstract holistic level, meaning we can abstract away hardware details of the implemented software functional unit by applying an evaluation and verification approach from a pure software point of view.

Based on the MARTE profile, many other applications also have been modeled for real-time embedded systems. For example, in [10], Andrél et al. used UML MARTE to represent the various IP-XACT (electronic component design standard, see [96]) figures, and through such an approach, electronic components and designs can be integrated into further system design and further modeling of component composition is facilitated.

The modeling and validation of timing constraints for real-time system is introduced already in the 1980s and 1990s in [6] and [1]. Abadi and Lamport imported variables for the examination of real-time context in temporal logic [1]. A state region graph technique is introduced by Alur, Courcoubetis and Dill in [6] for efficient automatic model-checking of real-time systems. A lot of work is devoted to the understanding of theoretical comprehension for timed automata. Those works include determination [165], minimization [163], power of clocks [164, 89], various extensions [56, 91, 48, 17] as well as logical characterizations [199, 91]. The famous UPPAAL tool, introduced in Section *3.5*, uses an extended timed automaton for specification of real-time system, and applies the state region graph technique for model checking. UPPAAL also solves the state explosion problem using compositional and symbolic model checking techniques [132]. We have chosen timed automata for the modeling and verification of real-time properties for reactive systems and use UPPAAL as the tool for real-time property verification. Comparing other approaches, e.g., the Communicating Sequential Processes (CSP) used in [9], UPPAAL has longer history and matured tools and open APIs. Other formalisms for real-time systems include Petri nets, timed process algebras, and real-time logic [23, 167, 198, 206]. Meanwhile, a number of other model checking tools for real-time systems are available including HYTECH [88] and KRONOS [54].

In [187], the authors study the modeling and verification of real-time systems as well as compositional specification for real-time systems based extensively on timed process algebra. The timed process algebra approach provides another alternative for hierarchical real-time system verification.

In [68], the authors describe a Domain Specific Modeling (DSL) language for distributed microcontroller applications. In particular, the model-driven approach can be applied to the exhaustive simulation of system configurations such that the adaptive firmware of embedded systems can be tested for runtime adaptation. Real-Time Maude [150] is a language and tool supporting the formal specification and analysis of real-time and hybrid systems. It was developed and is maintained at the Precise Modeling and Analysis (PMA) group at SINTEF ICT and University of Oslo in Norway. It has been used to model and analyze sophisticated communication protocols and scheduling algorithms as well as a semantic framework and formal analysis tool for a number of modeling languages for embedded systems.

In [67], Feng et al. proposed a novel approach for embedded real-time systems, i.e., a machine learning approach to generate models of real-time embedded systems. This approach applies learning algorithms to execution traces of black box components of real-time embedded systems so that the system level control flow model could be generated. This approach demonstrates more practical applicability for industrial applications.

The model checking of real-time properties for compositional systems are implemented in the UPPAAL tool [131] and the CMC tool [129]. In [204], Jianhua et al. analyzed the linear duration properties and designed an algorithm for model checking for real-time property of parallel composed real-time systems.

The probabilistic real-time properties of reactive systems usually describe the system safety aspect. In [137], Lu Feng et al. proposed a novel learning technique, based on L* [11], for generating probabilistic assumptions for probabilistic systems.

5 Probabilistic Model Verification

In recent years, immense attention has been attracted to probabilistic timed automata (PTAs) and probabilistic system behavior verification. The PRISM model verification tool [120] (see Section 3.5) gains considerable popularity for the combination of probability and timed behaviors in real-time and embedded system and networked protocols. In [85], Arnd Hartmanns and Holger Hermanns use the *digital clocks* approach for model checking of probabilistic timed automata and integrated the PRISM model checking engine into the compositional modelling language Modest [34]. As mentioned above, we applied the reachability based checking for the model checking of probabilistic real-time behaviors of compositional control systems modelled in Reactive Blocks. This helps to verify the probabilistic safety properties in our application of Reactive Blocks to cyber-physical systems.

6 Performance Modelling and Analysis

Modern distributed concurrent systems present Quality of Service (QoS) as an important requirement. Such *QoS* requirements, including performance, availability and reliability, are of crucial importance. This is especially true, if modern network protocols are the communication means to coordinate embedded control systems or to gather real-time data from widely distributed sensors. The analysis of systems presenting stochastic behaviors are often based on Stochastic Petri net models and other Petri net based models. In the following, we briefly mention some theories and tools for analyzing the *QoS* properties:

Stochastic Petri Net Models

Stochastic Petri nets (SPNs) [139] are widely used for performability and dependability evaluations. The specification of SPN is similar to Petri Nets except that tokens are fired using a stochastic time distribution. When using an SPN specification technique, one has to define a set of places P, a set of transitions T, and a set A of arcs between transitions and places or vice versa: $A \subseteq (P \times T) \cup (T \times P)$. Each place can contain zero or more tokens. Graphically, places are depicted as circles, transitions as bars, tokens as dots inside circles, and arcs as arrows.

The distribution of tokens over the places is called a marking and corresponds to the notion of *state* in a Markov chain [148]. Places from which arcs go to a transition are called input places of that transition. Output places are those, to which arcs come from a particular transition. A transition is said to be enabled when all of its input places contain at least one token. An enabled transition may fire. After firing, a transition removes one token from all of its input places and puts one token in all of its output places. Thus, a change of state occurs.

The firing of transitions is assumed to take an exponentially distribution time. Given the initial marking of an SPN, all the markings as well as the transition rates can be derived, and the number of tokens in every place is bound. Thus a finite Markov chain is obtained [191]. The reward rates are described as a function of the markings and the reward rates and the Markov chain together yield a Markov Reward Model [200] which captures some performance measure of interest. A list of SPN based tools for performance and reliability analysis is given in [192].

Other Extensions of Petri Net (PNs) Based Model

Different extensions to ordinary PNs have been developed in order to increase the modeling convenience and/or the modeling power. Colored PNs (CPNs), introduced by Jensen [103], are one such extension. The latter allows a type (color) to be attached to a token. A color function C assigns a set of colors to each place, specifying the types of tokens that can reside in the place. In addition to introducing token colors, CPNs also allow transitions to fire in different modes (transition colors). The color function C assigns a set of modes to each transition and incidence functions are defined on a per mode basis.

Generalized Stochastic PNs (GSPNs) allow us to use two types of transitions, i.e., immediate and timed. Once enabled, immediate transitions fire in zero time. If several immediate transitions are enabled at the same time, the next transition to fire is chosen based on firing weights (probabilities) assigned to the transitions [15].

Performance Evaluation for Real-Time Java

Our work supports the following approaches for real-time software performance evaluation. Since the present platform and code generation support the Java programming language, there is a necessity to do a survey on Java performance evaluation and prediction.

Benchmarking is an important way for evaluating performance of real-time systems. Many benchmarking tools are applied to the evaluation of real-time embedded systems. For instance, in [51], Corsaro et al. applied the RTJPerf benchmarking suite to evaluate the efficiency and predictability of several implementations of the Real-time Specification for Java (RTSJ). The results indicate that real-time Java is maturing to the point where it can be applied to certain types of real-time applications. PACE (Performance Analysis Characterization Environment) is an environment for characterizing and predicting the performance of distributed Java applications within dynamic heterogeneous environments [149]. It is based on a set of abstractions, formalization techniques, and the extraction of useful information from the Java application. In particular, the PACE approach applies timing analysis on Java byte codes so as to develop a method for predicting the performance of Java programs. The performance evaluation and prediction is performed on the byte codes level, which can accurately predict the JVM (Java Virtual Machine) performance. The analysis unit is referred to as the *Sequential Bytecode Block*.

Simulation and Emulation Tool for Networked Reactive Systems

There are plenty of network simulation tools available for the research of communication and computer networks. Since our approach exemplified a network environment in an 802.11 WLAN network for mobile embedded systems, we mainly surveyed some WLAN simulation and emulation tools for IEEE 802.11 wireless networks. Network simulators, including ns2 [99], OPNET [45], and NCTUns [197] can describe the state of the network (nodes,routers, switches, links) and network events (data transmissions, packet error etc.). Experimental research using these tools shows that the performance of a tele-operated robot system is strongly influenced by the quality of the communication environment. We take advantage of the simulation tools to analyze the saturation effect of a network, and the analysis results are used as input to our networked device safety analysis. For example, it is shown in [188] that the position tracking and control of robots degrades if the available bandwidth in the communication medium is occupied by too many stations. In [42], the benefits of the extensive use of wireless technologies in automation and robotics are discussed. The IEEE 802.11 series of wireless protocols has several flavors that can be applied to industrial robotics, of which IEEE 802.11a is considered to be the most suitable. An extensive survey on wireless sensor network emulators and simulators is discussed in [97].

7 Cyber-physical Systems

A cyber-physical system is an extension of a real-time and hybrid systems. With the growing popularity of CPS applications in IoT, smart cities, smart grids, industrial Internet and smart "everything", the research about CPS has become of great importance. In [161], Radhakisan Baheti and Helen Gill state that for analyzing the system level properties of CPS, "Standardized abstractions and architectures that permit modular design and development of cyber-physical systems are urgently needed". Also, due to the lack of such kind of architectures, a serious problem is exposed in verifying the overall correctness and safety of design of CPS at system level.

We give an answer to the cornerstone for the architecture of CPS by the Reactive Blocks approach provided by Kraemer and Herrmann, and we searched for the solution for analyzing the properties of CPS at system level.

Recent trends for dealing with the complexity of cyber-physical or IoT systems indicate that models should be able to capture the runtime behavior of complex systems. This requires the ability to abstract and describe self-adaptiveness or autonomous properties of software systems. We recommend to use Reactive Blocks for the component-based cyber-physical system development since it provides infrastructure for developing self-adaptive and autonomous systems in a model-driven manner. In this paradigm, we introspect Reactive Blocks as an execution model, and a dynamic component-based structure can be the dynamic model of cyber-physical systems. More details for Models@Runtime and how the Reactive Blocks approach can be used for runtime behavior modeling are discussed in Chapter 7.

Cyber-physical System Safety

Cyber-physical system safety is now one of the most urgent required properties which is recognized by the research community, the industry, as well as the International Society of Automation (ISA). ISA identifies the safety standards for automation systems in ISA84 (IEC 61511, [12]), works on aligning safety and security standards including ISA84 and ISA99 (IEC 62443, [13]). In [168], Giedre et al. proposed an approach for aligning CPS safety and security in early development phases by synchronizing safety and security life-cycles based on standards. The alignment is done by a unified model called Failure-Attack-CounTermeasure graph (FACT).

In [29], Blech and Schmidt proposed a framework for the verification of the spatial behavior of distributed software systems. At present, a tool called BeSpaceD has been developed for spatial behavior abstraction, and properties like range coverage and the absence of collisions can be studied. Ensuring spatial safety in hybrid and embedded control systems is a new research trend in both software verification and embedded systems. These systems are usually large and comprise many sub-systems, including sensors, communication networks, and controllers. With growing system sizes and the combination of independently developed systems into large systems or interplay of different subsystems, the safety aspect of such systems becomes very important. In [26], Blech et al., extend the verification of cyber-physical systems to a component level that describes the so called *behavioral types*. Further, the type of verification techniques are extended to verify spatial behavior types and put into an infrastructure that helps to design largely distributed collaboration parts [31]. Such kind of complex systems needs the joint effect of simulation, validation and visualization of cyber-physical systems.

The safety specification of CPS related with time and space behavior is also analyzed in [174], which gives another formal abstraction of spatial and time behaviors. In this paper, Spichkova et al. use the $FOCUS^{ST}$ language, which is an extension of the FOCUS specification language [38] to describe the Safety-Critical System properties. The FOCUS language is a special type of timed automata that is named Timed State Transition Diagrams (TSTDs). And the properties can be translated into a Higher-Order Logic and can be verified by the interactive semi-automatic theorem prover Isabelle [173, 147].

6 SUMMARY OF THE PAPERS

Seven papers are included into this thesis, and this chapter presents a brief summary of each one, including the individual contribution of each paper to the overall analysis method and Complex System Engineering. To make an overall comparison, I labeled each paper with its contribution to each of the four aspects in the Reactive Blocks method discussed in Chapter *1*, Section 2 and to overall Complex System Engineering as discussed in Chapter *1*, Section 1.

1 Paper A

Fenglin Han, Surya Bahadur Kathayat, Hien Le, Rolv Bræk, Peter Herrmann, *Towards Choreography Model Transformation via Graph Transformation*, Proceedings of the 2nd IEEE International Conference on Software Engineering and Service Sciences (ICSESS 2011). Beijing, $15-17^{th}$ **July 2011.**

This paper presents the first application of graph grammar techniques for system synthesis and analysis. We proposed a framework for the decomposition of system specifications in two levels of abstraction. The choreography model is a more abstract level specification for capturing the system behavior from a service requirement requisition. It is used to describe collaborative entities that are described with respect to their collaborative roles, and their interacting sub-services that are modeled by a customized UML activities. Reactive Blocks is used as the design time model for implementing service systems. In particular, these two models are characterized in the following manner:

- UML collaboration diagram is used to define the structure of a collaborative service, with service participants and sub-services are defined as collaboration uses. the ordering of collaboration uses are defined by activity diagrams called the flow-global choreography model, which specifies high-level global behavior including ordering and causality in the sub-services.
- Reactive Blocks model is used as a flow-localized choreography model for design-level system specification, such that systems can be directly implemented and their code can be generated.

Flow-global choreographies can be transformed into flow-localized choreographies using graph transformation techniques. I defined a set of transformation rules to implement the localization policies. The approach is highlighted by means of a case study featuring the European Rail Traffic Management System (ERTMS).

Unique Contributions of the author of this thesis include:

- Import of the graph transformation approach into the model-based system development synthesis and composition. Specifically, I applied the attributed graph grammar (AGG) system, which automatically connects existing component models in the library with new system developments, to system decomposition and derivation.
- I contribute to the overall approach of the paper and developed the transformation rules of the case study.

The author of the thesis contributes to 65% to this paper. This paper contributed to automation support from requirement modeling to functional modeling and derivation of functional components in applying Reactive Blocks approach to Complex System Engineering [1].

Structure aspect	**Transformational aspect**	**Compositional aspect**	**Analysis aspect**
$\star$	$\star\star\star\star\star$	$\star$	$\star\star$
Contribution to CSE: AS_1			

2 Paper B

Fenglin Han and Peter Herrmann, *Remedy of mixed initiative conflicts in model-based system engineering*, Proceedings of the 11th International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2012), Volume 47 of of the Electronic Communications of the EASST, 2012.

This paper introduces a typical application domain for graph transformation rules that can be used for automatic pattern maintenance and weaving for model based system development. In collaborative communication, often two participants are able to instantiate a certain cooperation. Here, a so-called mixed initiative conflict may arise when both participants initiate communication at the same time. This conflict may easily lead to unexpected and unwanted behavior if no precautions are taken. Using our Reactive Blocks-based incremental system development, a logical block was formalized to coordinate communication and prevent such conflicts. A basic policy is to have one side yielding to the other when a conflict occurs. In this paper, an automatic application of logical building blocks between conflicting participants (two communicating software entities) was proposed, by applying graph transformation rules to automatically discover mixed-initiative conflicts between communicating parties. Further, we apply graph transformation to automatically insert resolution policies into the relevant communications in the form of a logical pattern block. Using such procedures, automated techniques were applied to remedy human design errors. This paper illustrates an example of automation in software engineering, specifically regarding how erroneous design patterns can be discovered and solved using the graph transformation rule. We thank Vidar Slåtten for his button game example, which we used as an illustrative scenario in the paper.

Unique Contributions of the author of this thesis include:

- Integration of the graph transformation approach into the system analysis, to ensure that certain erroneous patterns are automatically discovered and immediately remedied.
- Contribution to the main approach of the paper and developed the transformation rules of the case study.

The author of the thesis contributes to 80% of the paper.

Structure aspect	**Transformational aspect**	**Compositional aspect**	**Analysis aspect**
$\star$	$\star\star\star\star\star$	$\star$	$\star\star\star$
Contribution to CSE:AS_1, AS_2			

3 Paper C

Fenglin Han, Peter Herrmann and Hien Le, *Modeling and Verifying Real-time Properties of Reactive Systems*, Proceedings of the 18th International Conference on Engineering of Complex Computer Systems (ICECCS 2013). Singapore, 2013.

In this paper, our model-transformation-based model analysis and verification approach was extended to a time-critical software system. In particular, we defined the core concept of the Real-Time External

[1]Related aspects are discussed in Section 2 of Chapter *1*

State Machine (RTESM). Based on the Reactive Blocks paradigm for incremental system development and verification, an RTESM is an extension of the External State Machine (ESM). It is a rich-label version containing clock labels, synchronization channels, and guard conditions. Thus, the RTESM describes a real-time contract between the internally implemented building blocks behavior and the externally visualized behavior. It follows the timed automata formalism and can be imported into the well-known UPPAAL tool for time-related verification by model transformation. The properties are described by a variation of computation tree logic called Timed Computation Tree Logic (TCTL). In this paper, we use a component designed to protect en electrical motor controller system against excessive speed as an introductory example.

Unique Contributions of the author of this thesis include:

- Extension of the Reactive Blocks model-driven development approach by the real-time external state machine (RTESM), to enable the real-time formalism and analysis of the Reactive Blocks model.
- Development of the infrastructure for the analysis and verification of the real-time Reactive Blocks model.
- Contribution to the development and delineation of the transformation approach, the design of the verification property, and its implementation.

The author of the thesis contributes to 80% of the paper.

Structure aspect	**Transformational aspect**	**Compositional aspect**	**Analysis aspect**
⋆⋆⋆⋆⋆	⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆
Contribution to CSE:AS_3			

4 Paper D

Fenglin Han and Peter Herrmann, *Modeling Real-Time System Performance with Respect to Scheduling Analysis*, Proceedings of the 2013 International Joint Conference on Awareness Science and Technology and Ubi-Media Computing (iCAST & UMEDIA 2013). Aizu city, Japan, 2013.

The analysis of real-time systems is complex and non-trivial, since it is highly associated with the underlying hardware platforms. Thus, model-driven development requires mechanisms to associate models with underlying implementation platforms. In this paper, we bridge the gap between the software model and the hardware by introducing the well-known UML profiling mechanism, that is, we present a performance profile for the Reactive Blocks paradigm. We annotated the subcomponents and actions of each building block with labels that identify common concepts in real-time performance analysis, including worst-case execution time (*wcet*), best case execution time (*bcet*), work load, periodic tasks, scheduling policies, and the computation time probability density distribution function (*ctddf*). By introducing these annotations, we provide a global view of real-time system performance analysis. This work also enables the scheduling analysis of a real-time critical software system by introducing timed automata that simulate resources and scheduling policies, mechanisms that are typically hidden from software developers and library users.

Unique Contributions of the author of this thesis include:

- Introduction of performance annotations for Reactive Blocks and connection of the building blocks to the performance issues by introducing the UML profiling mechanism.
- Enabling scheduling analysis of real-time Reactive Blocks by introducing automata that simulate resources and scheduling policies. Specifically, we made the real-time verification of the building blocks possible.

The author of the thesis contributes to 95% of the paper.

Structure aspect	Transformational aspect	Compositional aspect	Analysis aspect
⋆	⋆	⋆⋆	⋆⋆⋆⋆

Contribution to CSE: AS_3

5 Paper E

Fenglin Han, Jan Olaf Blech, Peter Herrmann and Heinz Schmidt. ***Towards Verifying Safety Properties of Real-Time Probabilistic Systems*****. Proceedings of the 11th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA 2014), Electronic Proceedings in Theoretical Computer Science 147, pp. 1–15. Grenoble, France. April** 12^{th}**, 2014.**

In this paper, I explore real-time and probabilistic properties of real-time systems. Due to maintenance costs, many embedded control systems require quantitative timing constraints that may include guaranteed probabilities for time and space. For example, a robot may be required to process a task in a predefined amount of time with a probability of 99.999999% to prevent damage to other equipment. I propose a framework for integrating probabilistic real-time verification and performance prediction with system development. Using the spatiotemporal analysis tool BeSpaceD that was integrated into Reactive Blocks, one can verify probabilistic spatial behaviors. This approach can be summarized as follows:

- One models embedded control systems with Reactive Blocks. The working environment of the embedded system, for instance the working path of a transportation robot is generated with simulation blocks.
- One annotates the core control function block with probabilistic real-time constraints.
- In the extended formalism, *probabilistic real-time* behavior is represented by a probabilistic real-time external state machine (PRTESM). It is transformed to probabilistic timed automata and imported to analyzing tools such as PRISM.
- Using the simulation data, one applies BeSpaceD for spatial verification. Specifically, one can generate the traces capturing spatiotemporal behavior, and input the trace data into the BeSpaceD tool.
- The core control building blocks are extracted and used for code generation following the analysis and simulation using the above approach, since both its functional and non-functional properties have been verified.

This paper use a transportation robot in a storage hall as an example to illustrate this approach.

Unique Contributions of the author of this thesis include:

- The author introduced of the formalism for the non-functional system property analysis, the probabilistic real-time external state machine (PRTESM). Also, based on RTESM extended the modeling capability to include probabilities and real-time attributes.
- The author introduced the new software verification tool, the PRISM tool for probabilistic real-time system analysis. In the new paradigm, system properties may be formalized using probabilistic computation tree logic (PCTL).

The author of the thesis contributes to 50% of the paper.

Structure aspect	Transformational aspect	Compositional aspect	Analysis aspect
	⋆	⋆	⋆⋆⋆⋆

Contribution to CSE:AS_3, AS_4

6 Paper F

Peter Herrmann , Jan Olaf Blech, Fenglin Han and Heinz Schmidt, ***A Model-based Tool-chain to Verify Spatial Behavior of Cyber-Physical Systems. In International Journal of Web Services Research (IJWSR)*****, Volume 13, No. 1 (2016).**

The 2014 Asia-Pacific Services Computing Conference (APSCC2014), Dec 2014, Fuzhou, China, awarded the best track paper award to an earlier version of this paper.

In this paper, we introduce a framework to preserve safety properties in cyber-physical systems. The proposed approach combines a Reactive Blocks model-driven development framework and spatiotemporal verification techniques. First, we use Reactive Blocks to develop the control software and to simulate the safety critical control systems. Thereafter, we examine the real-time verification techniques applied in the Reactive Blocks model, using the BeSpaceD tool to verify the spatial properties. BeSpaceD is combined in a way that is different to the one used in Paper E. That allows us to analyze the overall behavior of a Reactive Blocks model for spatiotemporal properties and not only certain simulated traces.

The approach can be summarized as follows:

- **Step 1**: Describe the physical properties of a cyber-physical component in BeSpaceD terms or generate BeSpaceD terms from Reactive Blocks models by writing Scala programs.
- **Step 2**: Model the control system using Reactive Blocks to simulate continuous behavior, by importing the physical properties from Step 1. These control system building blocks are checked against design errors by the Reactive Blocks internal model checkers.
- **Step 3**: Using the generated BeSpaceD syntax, verify the spatial behaviors in the composed tool; for example, verify collision avoidance.
- **Step 4**: Using the physical properties and reactive models of the above steps, check the real-time properties using the extended abstract model, the real-time external state machine (RTESM), as verified by the UPPAAL tool.
- **Step 5**: Transform the Reactive Blocks of the control systems into executable code for the controller of the real-life embedded system.

Unique Contributions of the author of this thesis include: Contribution to the real-time verification, and evaluated the real-time ESM (RTESM) for real-time property formalization and verification. Establishment of a master project to develop the extension to the Arctis tool set, in order to integrate the BeSpaceD and Arctis tools (supervised by *Professor* Peter Herrmann). The author of the thesis contributes to 40% of the paper.

Structure aspect	**Transformational aspect**	**Compositional aspect**	**Analysis aspect**
		⋆	⋆⋆⋆⋆
Contribution to CSE:AS_3, AS_4			

7 Paper G

Fenglin Han, Jan Olaf Blech, Peter Herrmann and Heinz Schmidt, ***Model-based Engineering and Analysis of Space-aware Systems Communicating via IEEE 802.11*****, In 39th Annual International Computers, Software & Applications Conference (COMPSAC 2015). pages 638-646, Taichung, Taiwan, July 2015, IEEE Computer.**

With our model-driven development framework and its supporting tool chain, UPPAAL and BeSpaceD, we completed a case study in hybrid control system development, analysis and verification. In this paper, we target the embedded control systems of mobile fulfillment robot systems that communicate in a wireless environment. Specifically, we use our model-driven framework for embedded control system modeling, and then analyze and simulate wireless network coordination. The analysis and simulation of the wireless network supplements the verification of the spatial conditions and ranges of the access points. We also import the simulation tool, Jemula, to generate network communication simulation data to achieve a virtual robot fulfillment system in a storage hall.

The concrete steps of the case study are done out as follows:

- First, we use constraint solutions to provide parameters for the network simulations. Spatiotemporal properties of a coordinated mobile fulfillment system are the first constraints established by the BeSpaceD tool since the physical layout of the access points in a WLAN affects the control accuracy of a robot. These constraints can drive the maximum free space distance between an access point and the moving robot (the moving speed of the robot can also be considered).
- We then use the results of the network simulations, including the failure probabilities and other statistical properties, to input further constraints into the constraint solver.
- The 802.11 WLAN protocol is used as the communication and coordination media, and we then analyze the control system models behavior in the given environment.

Unique Contributions of the author of this thesis include: Conducting the 802.11 WLAN analysis and simulation. Specifically, I applied an open source tool, Jemula802 [4], to configure an 802.11 WLAN network. The configuration details included free space distance, network traffic with packets at time intervals following a negative exponential distribution, and robot numbers. I obtained the maximum delay possibilities and provided the probabilities and maximum delay to the spatiotemporal property analysis tool, BeSpaceD. The author of the thesis contributes to 60% of the paper.

Structure aspect	**Transformational aspect**	**Compositional aspect**	**Analysis aspect**
			⋆⋆⋆
Contribution to CSE: AS_3, AS_4			

7 FUTURE WORK

This thesis opens analysis and research topics ranging from novel software modeling and synthesis to software verification techniques. In this section, we discuss a few ideas for future research.

1 Modeling and Approach Validation

The approach for extending the modeling capability of Reactive Blocks with annotations of clocks and guard conditions to represent real-time requirements is still under research and evaluation. More modeling and evaluation cases are needed to address the following issues:

- Modeling capability: Currently, we are using reactive models for generating software code that can simulate or emulate real-time behavior for application-specific needs. This approach needs more cases, especially in robotics and embedded systems.
- Analysis capability: Our work shows the extensibility of the SPACE method in more basic case studies. The analysis capability of this extended approach needs to be explored further in order to find out more about its scalability potentials.
- Combination with reliability: Reliability of software systems is a major topic for electronic and embedded systems. In our group, Vidar Slåtten has addressed this topic in his Ph.D. thesis for reliability conservation and mechanism extension in [179]. However, his work does not consider real-time issues in conjunction with software reliability due to the complexity of the problem. Future work would integrate the real-time analysis capability of this thesis with Slåtten's reliability analysis for component-based software development.

2 Taking Hardware into Consideration

The automata paradigm needs more concrete assumptions for the underlying operating systems and hardware. These hardware systems include circuit buses, cache blocks, and communication networks. In the future, the automata-based verification should be extended with rich definitions for environmental properties. This is the basis for more realistic software system analysis.

In addition, the best use of Reactive Blocks should be in the embedded software industry, in which real-time and performance evaluation depends in large part on the hardware. This requires broader platform support from our approach, including code generators for C, C++, and hardware-oriented programming languages, such as VHSIC hardware description language (VHDL) or Verilog.

Real-Time programming languages are needed to support access to clocks, delays, timeouts, deadline specifications, and scheduling. Hardware clocks are usually implemented by OS level services. Comparing the Reactive Block paradigm with VHDL leads to a lot of similarities between the two languages. Table 1 compares the programming language concepts supported by VHDL with those supported by Reactive Blocks. It tells us that Reactive Blocks not only supports all of the concepts that are necessary for a low-level language, but also supports advanced formal method-based analysis of component level properties, which are helpful ingredients in software-hardware codesign as well.

Table 1. A comparison of Reactive Block paradigm and VHDL language

VHDL	Reactive Blocks	Description
Entity	External State Machine (ESM)	Entity representing the interface specification (I/O) of the component
		ESM describes both I/O stimulus and the casual order of I/O
Architecture	Internal Behavior (UML activities)	describes the internal implementation of an entity / ESM
Configuration	Block pins, Block Parameters	Allow to specify different configuration for a single component
Attributes	Attribute	Additional attributes or parameter
Process	Reference Block	Component behavior description

From Table 1 we see that any low-level language for implementing hardware-specific features can be supported in Reactive Blocks.

3 Supporting Mechanisms for Real-Time

Real-time software systems belong to a special category of software and hardware that places a higher demand on time constraints. Usually real-time systems require support from underlying hardware mechanisms or have optimized application-specific hardware design criteria. The first level of support is the programming language, which includes support for interruption of actions, special scheduling algorithms, real-time garbage collection, and thread execution priorities. The Java community has introduced a specification for Real-time Java, JSR001, and a set of implementations of the resulting real-time specification have emerged, including IBM's WebSphere Real Time and Sun Microsystem's Java Real-Time System. In the current reactive models, the implementation and scheduling of state machine transitions are based on JavaFrame [86, 186] and the traditional Java programming language; yet, real-time constraints on Reactive Blocks requires novel underlying infrastructures, including real-time state machines, programming languages, and operating systems. After surveying the alternatives, we suggest some extensions:

PERC

Atego$^{\text{TM}}$ is an example of a commercially available real-time embedded virtual machine. Instead of using the real- time specification for Java (RTSJ), the Atego $Perc^{TM}$ virtual machine technology and tool chain is built using only Java Standard Edition (JSE). Perc is not fully compliant with RTSJ (Real-Time Specification for Java) since it targets at some specific hardware on commercial use.

LJRT

Lund Java-based Real-Time (LJRT) is a Java-to-C translator that enables the use of Java source code for real-time execution on machines without JVM. LJRT applications run on J2SE 1.5+, but LJRT does not constitute a $Java^{TM}$ platform. Consider building a bridge between Java and C or providing direct C code generation to building blocks brings more time preserving possibilities to Reactive Blocks, but since the Virtual Machines are becoming more and more popular, and as programmers are more and more accustomed to write once and run everywhere, this might not be the wise direction for extending building blocks.

Jamaica Virtual Machine

The Jamaica VM implements the Realtime Specification for Java (RTSJ) and support multiple platforms.

4 New Emerging Modeling Trends

A recent trend for managing the complexity of cyber-physical or IoT systems is that models should be able to capture the runtime behavior of complex systems. Doing this, requires the modeling of the self-adaptive or autonomous properties of software systems [155]. We explored some models and modeling approaches, especially recent emerging modeling trends like Model@Runtime [18]. Compared with declarative modeling techniques, Models@Runtime is designed to extend the applicability of models and abstractions to the runtime environment. As its name indicates, Model@Runtime models the runtime environment, including time, memory, energy, location, platform, etc. We discover that Reactive Blocks and component-based cyber-physical system development provide the infrastructure for developing self-adaptive and autonomous systems in a model-driven manner, and thus further provide a way for modeling runtime behavior. In this paradigm, we introspect about using the Reactive Blocks as an execution model which reveals the essential logic of each modules of a distributed syber-physical system, and an artifect built on top of a runtime plugin system can be the dynamic model of cyber-physical systems. Existing technologies, such as the OSGi Framework [154] and Kevoree environment [69], already support the dynamic structure of runtime models. We address the relation between Reactive Blocks and model@Runtime using the five dimensions for classification of modeling approaches that are discussed in [18].

- Runtime model: What is the runtime model? The underlying execution model of the runtime model should be Reactive Blocks; that is, customized UML activities and state machines are the execution model. A dynamic model should be created supporting the existing dynamic component standard, OSGi. Thus, two levels of modeling hierarchies would be needed for model@Runtime.
- Purpose: What is its purpose? The Reactive Blocks approach provides a development and component execution model and reusable libraries for developers to maximize software reusability. The Reactive Blocks approach also provides traceability and animates the execution trace [118].
- User: Developers use Reactive Blocks for component-level development in order to guarantee the correctness of any required component-level property analysis. With this strong analytical capability, Reactive Blocks can guarantee the correctness, reliability, and system performance of cyber-physical systems.
- Properties: What are its properties? Reactive Blocks provides an execution model for component-level system behavior and at the same time provides a strong back-end for component functional correctness, reliability, and extended formalism. It can even provide a system level safety analysis.
- Reflection: Is the runtime model used for reflection? Reactive Blocks can provide causal connection between models and systems.

In existing works, e.g., in [25], Blech defined a framework for the modeling of behavioral specification for OSGi [154] bundles. The behavioral specification can be used to check compatibility of bundles and desired interacting protocols in a finite state machine-based manner.

5 Conclusion

The development of internet and networked systems are inevitably towards a more and more autonomous, environment-aware, ubiquitous direction. During this process, great challenges are brought in front of software engineers and industrial practitioners. Reactive Blocks brought a visual way of presenting the complexity of reactive systems and further extending the complexities along as the system grows. This thesis takes a glance of those complexities and try to have some tentative attempts. The future is promising and can be fruitful in various aspects.

Bibliography

[1] Abadi M, Lamport L. An old-fashioned recipe for real time. Real-Time: Theory in Practice: REX Workshop Mook, The Netherlands, June 3–7, 1991 Proceedings. Springer Berlin Heidelberg. pages 1-27, 1992.

[2] Abadi M, Lamport L. The Existence of Refinement Mappings. Theoretical Computer Science, 82(2):253-284, May 1991.

[3] Aceto L, Bouyer P, Burguño A, and Larsen, K G. The power of reachability testing for timed automata. Theoretical Computer Science, 300(1-3):411-475, 2003.

[4] ACM. ACM TURING AWARD GOES TO PIONEER WHO ADVANCED RELIABILITY AND CONSISTENCY OF COMPUTING SYSTEMS. ACM news release addressing the Turing Award for 2013. link: http://www.acm.org/press-room/news-releases/2014/pdfs/turing-award-lt-13a.pdf

[5] Alur R, Courcoubetis C, and Dill D. Model checking in dense real time. Information and Computation, 1993.

[6] Alur R, Courcoubetis C, and Dill D L. Model-Checking for Real-Time Systems, In *5th Symposium on Logic in Computer Science (LICS90)*, 1990, pp. 414–425.

[7] Alur R and Henzinger T. Reactive modules. Formal Methods in System Design, 15(1):7–48, 1999.

[8] Andrade E, Maciel P, Callou G and Nogueira B. A Methodology for Mapping SysML Activity Diagram to Time Petri Net for Requirement Validation of Embedded Real-Time Systems with Energy Constraints. In proceeding of 2009 Third International Conference on Digital Society. pages 266-271. 2009.

[9] André A, Liu Y, Sun J, Dong J S. Parameter synthesis for hierarchical concurrent real-time systems. Journal of *Real-Time Systems*, pages 620–679. Nov. 01, 2014.

[10] Andrél C, Mallet F, Khan A M, and Simone R D. Modeling spirit ip-xact with uml marte.

[11] Angluin D, Learning regular sets from queries and counterexamples, Information and Computation, vol. 75, no. 2, pp. 87–106, 1987.

[12] ANSI/ISA 84.00.01-2004, Application of Safety Instrumented Systems for the Process Industries. The Instrumentation, Systems, and Automation Society, Research Triangle Park, NC (2004).

[13] ANSI/ISA-99-00-01-2007. Security for Industrial Automation and Control Systems. Part 1: Terminology, Concepts, and Models. The Instrumentation, Systems, and Automation Society, Research Triangle Park, NC (2007)

[14] Arendt T, Biermann E, Jurack S, Krause C, and Taentzer G. (2010) Henshin: Advanced concepts and tools for in-place emf model transformations. In Petriu, D., Rouquette, N., and Haugen, y., editors, *Model Driven Engineering Languages and Systems*, volume 6394 of *Lecture Notes in Computer Science*, pages 121–135. Springer Berlin Heidelberg.

[15] Bause F and Kritzinger F. Stochastic Petri Nets—An Introduction to the Theory, second ed. Vieweg Verlag, 2002.

[16] Beetz K and Böhm W. Challenges in engineering for software-intensive embedded systems. In Pohl, K., Hönninger, H., Achatz, R., and Broy, M., editors, *Model-Based Engineering of Embedded Systems*, pages 3–14. Springer Berlin Heidelberg (2012).

[17] Behrmann G, Fehnker A, Hune T, Larsen K G, Pettersson P, Romijn J, and Vaandrager F. Minimum-cost reachability for priced timed automata. In Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01), volume 2034 of Lecture Notes in Computer Science, pages 147–161. Springer, 2001.

[18] Bencomo N, France R, Götz S, Rumpe B. Summary on the 8th International Workshop on Models@run.time. In: Proceedings of the 8th Workshop on Models @ Run.time. Colocted with MODELS 2013, Miami, USA, pg 1-8, CEUR Workshop Proceedings, Vol-1079, September, 2013.

[19] Bengtsson J, Larsson F, Pettersson P, Wang Y, Christensen P, Jensen J, Jensen P, Larsen K, and Sorensen T. UPPAAL: A Tool Suite for Validation and Verification of Real-Time Systems, In *Hybrid Systems III*, LNCS 1066, pages 232–243, Springer-Verlag, 1996.

[20] Bengtsson J, and Wang Y. Timed automata: Semantics, algorithms and tools. In Desel, J., Reisig, W., and Rozenberg, G., editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer Berlin Heidelberg (2004).

[21] Berezin S. Model Checking and Theorem Proving:a Unified Framework, PhD thesis at School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213.

[22] Berlemann L, and Mangold S. Appendix A: Jemula802. *Cognitive Radio and Dynamic Spectrum Access*, John Wiley & Sons, 2009.

[23] Berthomieu B and Diaz M. Modeling and verification of timed dependent systems using timed petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273. 1991.

[24] Biermann E, Ehrig K, Ermel C, Köhler C, and Taentzer G. The EMF Model Transformation Framework, A. Schürr, M. Nagl, and A. Zündorf (Eds.): AGTIVE 2007, LNCS 5088, pp. 566–567, 2008.

[25] Blech J O. Towards a Framework for Behavioral Specifications of OSGi Components. Formal Engineering Approaches to Software Components and Architectures 2013 (FESCA'13) EPTCS 108, 2013, pp. 79–93.

[26] Blech J O, Herrmann P. Behavioral Types for Component-based Development of Cyber-Physical Systems. Workshop on Human-Oriented Formal Methods, 2015.

[27] Blech J O, Schätz B. Towards a Formal Foundation of Behavioral Types for UML State-Machines, ACM SIGSOFT Software Engineering Notes, Volume 37 Number 4. July 2012.

[28] Functional safety of electrical/electronic/ programmable electronic safety-related systems, 1998. International Electrotechnical Commission.

[29] Blech J O and Schmidt H W. BeSpaceD: Towards a Tool Framework and Methodology for the Specification and Verification of Spatial Behavior of Distributed Software Component Systems, Journal of CoRR, 2014.

[30] Blech J O, Schmidt H W. Towards modeling and checking the spatial and interaction behavior of widely distributed systems, Improving In Journal of Improving Systems and Software Engineering Conference, Melbourne, 2013

[31] Blech JO, Spichkova M, Peake I, Schmidt H W. Cyber-virtual systems: Simulation, validation & visualization 2014 International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), 2014.

[32] Boehm B. A View of 20th and 21st Century Software Engineering. Proceedings of the 28th International Conference on Software Engineering, (ICSE06), pages 12–29, 2006, Shanghai, China.

[33] Boehm, B. Some Future Trends and Implications for Systems and Software Engineering Processes, Systems Engineering, vol. 9, No. 1, 2006, pp 1–19.

[34] Bohnenkamp H C, D'Argenio P R, Hermanns H, and Katoen J P. "MoDeST: A compositional modeling formalism for hard and softly timed systems," IEEE Trans. Softw. Eng., vol. 32, no. 10, pp. 812–830, 2006

[35] Bouyer P. Timed Automata-From Theory to Implementation. LSV- CNRS & ENS de Cachan France. 2003.

[36] Bresciani P, and Giorgini P. The tropos analysis process as graph transformation system. In *In Proceedings of the Workshop on Agent-oriented methodologies, at OOPSLA 2002*, pages 1–12.

[37] Briand L, Morasca S, and Basili V. *"Property-based software engineering measurement,"* in IEEE Transactions on Software Engineering, vol. 22, no. 1, pp. 68-86, Jan 1996.

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

[39] Bruel, J M. Transforming Models with ATL. Satellite Events at the MoDELS 2005 Conference: MoDELS 2005 International Workshops Doctoral Symposium, Educators Symposium Montego Bay, Jamaica, October 2-7, 2005 Revised Selected Papers, 2006, Springer Berlin Heidelberg.

[40] Bruno B, Samuel B, Cornes C, Judicaël C, and Filliâtre JC, Eduardo G, Hugo H, et al. The Coq Proof Assistant Reference Manual : Version 6.1. Web link:https://hal.inria.fr/inria-00069968/file/RT-0203.pdf

[41] Bruno B, Sifakis J. Embedded Systems Design: The Artist Roadmap for Research and Development. Springer-Verlag Berlin Heidelberg, 2005. DOI:10.1007/b106761.

[42] Calcagno R, Rusina F, Deregibus F, Vincentelli A S , and Bonivento A. Application of Wireless Technologies in Automotive Production Systems. *VDI Berichte*, 1956:57–58, 2006.

[43] Castejon H, Braek R, and Von Bochmann G. Realizability of collaboration-based service specifications. In *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, pages 73 –80.

[44] CESAR (2010). http://www.cesarproject.eu/ Accessed Jan 2011.

[45] Chang X. Network simulations with OPNET. In *Proc. 31st Conference on Winter Simulation: Simulation — a Bridge to the Future (WSC'99)*, vol. 1, ACM, pages 307–314.

[46] Chikofsky E J, and J H. Cross. Reverse engineering and design recovery - a taxonomy. IEEE Software, pages 13–17, January 1990.

[47] Chiola G, Franceschinis G, Gaeta R, and Ribaudo M. GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets. Performance Evaluation, special issue on Performance Modeling Tools, 24(1 and 2):47–68, November 1995.

[48] Choffrut C and Goldwurm M. Timed automata with periodic clock constraints. Journal of Automata, Languages and Combinatorics (JALC), 5(4):371–404, 2000.

[49] Clarke E M, Grumberg O, and Peled D. Model checking. *MIT Press*, 1999.

[50] Corradini A, Montanari U, Rossi F, Ehrig H, Heckel R, Löwe M. Algebraic approaches to graph transformation part I: Basic concepts and double pushout approach. In G. Rozenberg, editor, Handbook of Graph Grammars and Computing by Graph transformation, Volume 1: Foundations, pages 163–246. World Scientific, 1997. 447, 448

[51] Corsaro A, Schmidt D C. In Proceedings of 8th Real-Time and Embedded Technology and Applications Symposium, 2002. pages 90-100.

[52] Dasarathy B. (1985). Timing constraints of real-time systems: Constructs for expressing them, methods of validating them. *Software Engineering, IEEE Transactions on*, SE-11(1):80–86.

[53] Daws C, Kwiatkowska M Z, and Norman G. "Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM," STTT, vol. 5, no. 2-3, pp. 221–236, 2004.

[54] Daws C, Olivero A, Tripakis S, and Yovine S. The tool KRONOS. In Proc. Hybrid Systems III: Verification and Control (1995), volume 1066 of Lecture Notes in Computer Science, pages 208–219. Springer, 1996.

[55] De Alfaro L. Formal verification of probabilistic systems, Ph.D. thesis, Stanford University (1997).

[56] Demichelis F and Zielonka W. Controlled timed automata. In Proc. 9th International Conference on Concurrency Theory (CONCUR'98), volume 1466 of Lecture Notes in Computer Science, pages 455–469. Springer, 1998.

[57] De Moura L, Bjørner N. Z3: An efficient SMT solver. In Tools and Algorithms for the Construction and Analysis of Systems (pp. 337-340). Springer, 2008.

[58] Depke R, Heckel R, and Malte Küster J. (2001). Agent-oriented modeling with graph transformation. In Ciancarini, P. and Wooldridge, M., editors, *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in Computer Science*, pages 105–119. Springer Berlin Heidelberg.

[59] D'Silva V, Kroening D, and Weissenbacher G. (2008). A survey of automated techniques for formal software verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(7):1165–1178.

[60] Eclipse Consortium. Eclipse Modeling Framework (EMF) – Version 2.2.0 (2006), DOI: http://www.eclipse.org/emf

[61] Eclipse Foundation. Henshin/ Graphical/ Editor, https://wiki.eclipse.org/Henshin_Graphical_Editor.

[62] Ehrig H, Heckel R, Korff M, Löwe M, Ribeiro L, Wagner A, and Corradini A. Algebraic approaches to graph transformation II: Single pushout approach and comparison with double pushout approach. In G. Rozenberg, editor, The Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations, pages 247–312. World Scientific, 1996. 447, 448

[63] Elmansouri R, Hamrouche H, Chaoui A. From UML Activity Diagrams to communication sequential processes (CSP) Expressions: A Graph Transformation Approach using Atom3 Tool. International Journal of Computer Science, 8(2), 2011.

[64] Eshuis R. Symbolic Model Checking of UML Activity Diagrams Journal ACM Transactions on Software Engineering and Methodology (TOSEM) TOSEM Homepage archive Volume 15 Issue 1, January 2006. Pages 1-38

[65] Fatima U. A Modular Method for High- Level Service Specification and Component Design Derivation, *Norwegian University of Science and Technology.* 2017.

[66] Fatima U, Braek R. "Modelling multiplicity in choreography models," 2013 3rd International Workshop on Model-Driven Requirements Engineering (MoDRE), Rio de Janeiro, 2013, pp. 74-78.

[67] Feng T, Wang L, Zheng W, Kanajan S, and Seshia S. (2007). Automatic model generation for black box real-time systems. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1–6.

[68] Fleurey F, Morin B, and Solberg A. (2011). A model-driven approach to develop adaptive firmwares. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, pages 168–177, New York, NY, USA. ACM.

[69] Fouquet F, Morin B, Fleurey F, et al. A dynamic component model for cyber physical systems. In Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering, pages 135–144. ACM, 2012.

[70] Fowler M. (1996-11-27). Analysis Patterns: Reusable Object Models. Addison-Wesley. ISBN 0-201-89542-0.

[71] France R and Bernhard R. Model-driven Development of Complex Software: A Research Roadmap, 2007 Future of Software Engineering. page. 37–54. IEEE Computer Society, Washington, DC, USA.

[72] Gamma E, Helm R, Johnson R and Vlissides J. DesignPatterns: Abstraction and Reuse of Object-Oriented Design. In:Nierstrasz,O.M. (Editor):7th European Conference on Object-Oriented Programming, Kaiserslautern,1993. Springer-Verlag.

[73] Garland S J and Lynch N. Using I/O Automata for Developing Distributed Systems. In Foundations of component-based systems, Gary T. Leavens and Murali Sitaraman (Eds.). Cambridge University Press, New York, NY, USA 285-312.

[74] Ge N, Pantel M and Zilio S D. Formal Verification of User-Level Real-Time Property Patterns. Eleventh International Symposium on Theoretical Aspects of Software Engineering. DOI: 10.1109/TASE.2017.8285630

[75] Ghamarian A H, de Mol M J, Rensink A, Zambon E, and Zimakova M V. (2010). Modelling and analysis using groove. Technical Report TR-CTIT-10-18, Centre for Telematics and Information Technology University of Twente, Enschede.

[76] Goranko V, Undecidability and Temporal Logic: Some Landmarks from Turing to the Present. Proceedings - 2012 19th International Symposium on Temporal Representation and Reasoning, TIME 2012.

[77] Grosskopf D and Weske. (Feb 28, 2009). The Process: Business Process Modeling using BPMN. Meghan Kiffer Press. ISBN 978-0-929652-26-9.

[78] Guellati S, Kitouni I, Saidouni D E. Verification of Durational Action Timed Automata using UPPAAL, International Journal of Computer Applications, pp. 33-41, LNCS 5596, Published by Foundation of Computer Science, New York, USA,October 2012.

[79] Gunawan, L A, Herrmann P. (2013) Compositional Verification of Application-Level Security Properties. Lecture Notes in Computer Science. vol. 7781.

[80] Gunawan L A, Herrmann P, Kraemer F A. (2009) Towards the Integration of Security Aspects into System Development Using Collaboration-Oriented Models. Communications in Computer and Information Science. vol. 58.

[81] Gunawan, L A, Kraemer F A, Herrmann P. A Tool-Supported Method for the Design and Implementation of Secure Distributed Applications. In Lecture Notes in Computer Science 6542, Springer Berlin Heidelberg. pages 142-155.

[82] Gunawan, L A, Kraemer F A, Herrmann P. (2012) Behavioral Singletons to Consistently Handle Global States of Security Patterns. Lecture Notes in Computer Science. vol. 7272.

[83] Hall A. Seven Myths of Formal Methods. IEEE Computer Society Press. *IEEE Software*, volume 7 pages 11-19. September 1990.

[84] Hansson H, and Jonsson B. A logic for reasoning about time and reliability. Formal Aspects of Computing, 6(4):512–535, 1994.

[85] Hartmanns A and Hermanns H. A Modest Approach to Checking Probabilistic Timed Automata, In Proc. Sixth International Conference on Quantitative Evaluation of Systems (QEST 2009), 13-16 September 2009, Budapest, Hungary, pages 187-196. September 2009.

[86] Haugen Ø, Møller-Pedersen B. JavaFrame – framework for java enabled modelling. *In: Proceedings of Ericsson Conference on Software Engineering*, (September). 2000.

[87] Henzinger T A. Symbolic model checking for real-time systems. Information and Computation, 111:193–244, 1994.

[88] Henzinger T A, Ho P H, and Howard W T. HYTECH: A model-checker for hybrid systems. Journal on Software Tools for Technology Transfer (STTT), 1(1–2):110–122, 1997

[89] Henzinger T A, Kopke P W, and Howard W T. The expressive power of clocks. In Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP'95), volume 944 of Lecture Notes in Computer Science, pages 417–428. Springer, 1995.

[90] Henzinger T, Nicollin X, Sifakis J, Yovine S. Symbolic model checking for real-time systems, Information and Computation 111 (2) (1994) 193–244.

[91] Henzinger T A, Raskin J F and Schobbens P Y. The regular realtime languages. In Proc. 25th International Colloquium on Automata, Languages and Programming (ICALP'98), volume 1443 of Lecture Notes in Computer Science, pages 580–591. Springer, 1998.

[92] Hermann F, Ehrig H, and Taentzer G. (2008). A typed attributed graph grammar with inheritance for the abstract syntax of uml class and sequence diagrams. *Electron. Notes Theor. Comput. Sci.*, 211:261–269.

[93] Herrmann C, Krahn H, Rumpe B, Schindler M, Völkel S. Scaling-Up Model-Based-Development for Large Heterogeneous Systems with Compositional Modeling. In: Proceedings of the 2009 International Conference on Software Engineeering in Research and Practice, Vol. 1. Ed.: H. Arabnia, H. Reza. July 13-16. Las Vegas, Nevada, USA.

[94] Herrmann P. Problemnaher korrektheitssichernder Entwurf von Hochleistungsprotokollen. Deutscher Universitätsverlag, 1998 (in German).

[95] Herrmann P, Krumm H. A Framework for Modeling Transfer Protocols. *Computer Networks*, 34(2) (2000) 317–337.

[96] IEEE Standard 1685-2014. IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows. 2014. doi:10.1109/IEEESTD.2014.6898803. ISBN 978-0-7381-9226-0.

[97] Imran M, Said A M and Hasbullah H. A Survey of Simulators, Emulators and Testbeds for Wireless Sensor Networks. In *International Symposium in Information Technology (ITSim)*, vol. 2, pages 897–902, 2010.

[98] Ingeol Chun, Kim J, Kim WT, Lee E. Self-Managed System Development Method for Cyber-Physical Systems. Control and Automation, and Energy System Engineering. Volume 256 of the series Communications in Computer and Information Science pp 191-194.

[99] Issaryakul T and Hossain E. *Introduction to Network Simulator NS2*. 2nd Edition, Springer-Verlag, 2012.

[100] Jagenberg T and Hunt J J. Real-time Blocks, Integrating Reactive Blocks with JamaicaVM. link: http://www.bitreactive.com/wp-content/uploads/2016/09/RealtimeBlocks-Integrating-Reactive-Blocks-with-JamaicaVM-2016-09.pdf, pages 94-103, Vienna, Austria, 2007.

[101] Jayaraman, P., Whittle, J., Elkhodary, A., and Gomaa, H. (2007). Model composition in product lines and feature interaction detection using critical pair analysis. In Engels, G., Opdyke, B., Schmidt, D., and Weil, F., editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *Lecture Notes in Computer Science*, pages 151–165. Springer Berlin / Heidelberg.

[102] Jensen H E. Model checking probabilistic real time systems. *In: Proc. 7th Nordic Workshop on Programming Theory.* pp. 247–261 (1996)

[103] Jensen K. Coloured Petri Nets and the Invariant Method, pp. 327-338. Math. Foundations on Computer Science, 1981.

[104] Jurdzinski M, Laroussinie F, and Sproston J. (2008). Model checking probabilistic timed automata with one or two clocks. *CoRR*, abs/0809.0060.

[105] Kastenberg H. (2008). PhD thesis, University of Twente, Enschede.

[106] Kathayat S B, On the Development of Situated Collaborative Services, PhD thesis, Norwegian University of Science and Technology (2012).

[107] Kathayat S B and Bræk R. (2010). From flow-global choreography to component types. In *System Analysis and Modeling (SAM)*, volume 6598 of *Lecture Notes in Computer Science*. Springer - Verlag.

[108] Kathayat S B, Hien N L and Bræk R. A Model-Driven Framework for Component-Based Development, SDL 2011: Integrating System and Software Modeling: 15th International SDL Forum Toulouse, France, July 5-7, 2011.

[109] Kerkouchea E, Chaouib A, Bourennanec E, Labbanic O. A UML and Colored Petri Nets Integrated Modeling and Analysis Approach using Graph Transformation. In: Journal of Object Technology, vol. 9, no. 4, pp. 25-43. JOT (2010)

[110] Kraemer F A. Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks. *Norwegian University of Science and Technology.* August, 2008.

[111]

[112] Kraemer F A, Bræk R, and Herrmann P. Compositional Service Engineering with Arctis. *Telektronikk*, 105(2009)1.

[113] Kraemer F A, Herrmann P. Transforming Collaborative Service Specifications into Efficiently Executable State Machines. *ECEASST* 6, 2007.

Kraemer F A, Herrmann P, Formalizing Collaboration-Oriented Service Specifications using Temporal Logic. In Proceedings of the Networking and Electronic Commerce Research Conference 2007 (NAEC), pages 194–220, Riva del Garda, ATSMA, October 2007.

[114] Kraemer F A, Herrmann P. Automated Encapsulation of UML Activities for Incremental Development and Verification In I nternational Conference on Model Driven Engineering, Languages and Systems. Springer.

[115] Kraemer F A, Herrmann P. Reactive Semantics for Distributed UML Activities. In J. Hatcliff, E. Zucca, Formal Techniques for Distributed Systems, Proceedings of the Joint 12th IFIP WG 6.1 International Conference (FMOODS 2010) and 30th IFIP WG 6.1 International Conference (FORTE 2010), Amsterdam, LNCS 6117, pages 17-31, Springer-Verlag, June 2010.

[116] Kraemer F A, Herrmann P, and Braek R. (2006b). Aligning UML 2.0 State Machines and Temporal Logic for the Efficient Execution of Services. In Meersman, R. and Tari, Z., editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4276 of *Lecture Notes in Computer Science*, pages 1613–1632. Springer Berlin Heidelberg.

[117] Kraemer F A, Slåtten V, Herrmann P. Engineering Support for UML Activities by Automated Model-Checking - An Example. *In Proceedings of the 4th International Workshop on Rapid Integration of Software Engineering Techniques* (RISE 2007), Luxemburg, November 2007.

[118] Kraemer F A, Slåtten V, and Herrmann P. Tool support for the rapid composition, analysis and implementation of reactive services. *Journal of Systems and Software*, 82(12):2068 – 2080 (2009).

[119] Krause C, Tichy M, and Giese H. (2014). Implementing graph transformations in the bulk synchronous parallel model. In Gnesi, S. and Rensink, A., editors, *Fundamental Approaches to Software Engineering*, volume 8411 of *Lecture Notes in Computer Science*, pages 325–339. Springer Berlin Heidelberg.

[120] Kwiatkowska M, Norman G, Parker D, PRISM: probabilistic symbolic model-checker, *in Lecture Notes in Computer Science, Computer Performance Evaluation.* Heidelberg, Germany:Springer-Verlag, 2002, vol. 2324, pp. 200–204.

[121] Kwiatkowska M, Norman G, Parker D. *PRISM 4.0: Verification of Probabilistic Real-time Systems.* Proc. 23rd International Conference on Computer Aided Verification (CAV'11), pp. 585–591, 2011.

[122] Kwiatkowska M Z, Norman G, Parker D and Sproston J, "Performance analysis of probabilistic timed automata using digital clocks," Formal Methods in System Design, vol. 29, no. 1, pp. 33–78, 2006.

[123] Kwiatkowska M, Norman G, Sproston J and Wang F. Symbolic Model Checking for Probabilistic Timed Automata. In Y. Lakhnech and S. Yovine (editors), Proceeding of FORMATS/FTRTFT'04, volume 3253 of Lecture Notes in Computer Science, pages 293-308, Springer. September 2004.

[124] Kwiatkowska M Z, Norman G, Sproston J, and Wang F, Symbolic Model Checking for Probabilistic Timed Automata, Information and Computation, vol. 205, no. 7, pp. 1027–1077, 2007.

[125] Lamport L. "Sometimes" is sometimes "not ever": A Tutorial on the Temporal Logic of Programs. In Proceedings of the Seventh Annual Symposium on Principles of Programming Languages (POPL), pages 174-185, ACM SIGSOFT-SIGPLAN, ACM, January 1980.

[126] Lamport L. (1994). The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923.

[127] Lamport L. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. *Addison-Wesley*, (2002).

[128] Lara J and Vangheluwe H. (2002). Atom3: A tool for multi-formalism and meta-modelling. In Kutsche, R.-D. and Weber, H., editors, *Fundamental Approaches to Software Engineering*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer Berlin / Heidelberg.

[129] Laroussinie F, Larsen K G. CMC: A Tool for Compositional Model-Checking of Real-Time Systems. Formal Description Techniques and Protocol Specification, Testing and Verification: FORTE XI/PSTV XVIII'98 IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XI) and Protocol Specification, Testing and Verification (PSTV XVIII) 3–6 November 1998, Paris, France. Springer US, 1998. In Meersmann, R. and Tari, Z., editors, *Proceedings of the 8th International Symposium on Distributed Objects and Applications (DOA), 2006, Montpellier, France*, volume 4276 of *Lecture Notes in Computer Science*, pages 1613–1632. Springer–Verlag Heidelberg.

[130] Laroussinie F, Markey N, and Schnoebelen Ph. Model Checking Timed Automata with One or Two Clocks. In *Concurrency Theory (CONCUR)*, LNCS 3170, pages 387–401, Springer-Verlag, 2004.

[131] Larsen K G, Pettersson P and Wang Y. Compositional and Symbolic Model-Checking of Real-Time systems. In Proc. of the 16^{th} IEEE Real-Time Systems Symposium, pages 76-87, December 1995.

[132] Larsen K G, Pettersson P and Wang Yi. Model-Checking for Real-Time Systems. In Proceedings of the 10th International Symposium on Fundamentals of Computation Theory (FCT '95), Horst Reichel (Ed.). Springer-Verlag, London, UK, page, 62-88.

[133] Le Berre D and Parrain A. The Sat4j library, release 2.2. Journal on Satisfiability, Boolean Modeling and Computation, Volume 7 (2010), system description, pages 59-64.

[134] Lengyel L, Levendovszky T, Mezei G and Charaf H. Model transformation with a visual control flow language. *International Journal of Computer Science (IJCS)*, 1(1):45–53.

[135] Lichtenstein O, Pnueli A. Checking that finite-state concurrent programs satisfy their linear specification. In Proceedings of the 12th Annual Symposium on Principles of Programming Languages, pages 97-107. ACM Press, 1985.

[136] López-Grao J P, Merseguer J, and Campos J. (2004). From uml activity diagrams to stochastic petri nets: Application to software performance engineering. In *Proceedings of the 4th International Workshop on Software and Performance*, WOSP '04, pages 25–36, New York, NY, USA. ACM.

[137] Lu F, Kwiatkowska M and Parker D. Compositional Verification of Probabilistic Systems using Learning. In Proc. 7th International Conference on Quantitative Evaluation of Systems (QEST'10), pages 133-142, IEEE CS Press. September 2010.

[138] Malewicz G, Austern M H, Bik A J, Dehnert J C, Horn I, Leiser N, Czajkowski G. Pregel: a system for large-scale graph processing. In: Proc. SIGMOD 2010, pp. 135–146. ACM (2010), doi:10.1145/1807167.1807184

[139] Marsan M. A. 1990. Stochastic Petri nets: an elementary introduction. In Advances in Petri nets 1989, Grzegorz Rozenberg (Ed.). Lecture Notes In Computer Science, Vol. 424. Springer-Verlag New York, Inc., New York, NY, USA 1-29.

[140] Martinez H N C. Collaborations in service engineering: modeling, analysis and execution. *Norwegian University of Science and Technology.* 2008.

[141] Merlin P M. A Methodology for the Design and Implementation of Communication Protocols. IEEE Transactions on Communications, 24(6):614-621, 1976.

[142] Merlin P M. Specification and Validation of Protocols. IEEE Transactions on Communications, 27(11):1671-1680, 1979.

[143] Mohagheghi P, Fernandez M, Martell J, Fritzsche M, Giliani W. MDE adoption in industry: Challenges and success criteria. In: ChaMDE 2008 Workshop Proceedings: International Workshop on Challenges in Model-Driven Software Engineering. (2008)

[144] Moser L E, Ramakrishna Y, Kutty G, Melliar-Smit P M and Dillon L K, A graphical environment for the design of concurrent real-time systems, ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 6, no. 1, pp. 31 79, 1997.

[145] Murphey Y L, Masrur M A, Chen ZH and Zhang B. Model-based fault diagnosis in electric drives using machine learning, In Journal of IEEE/ASME Transactions on Mechatronics, volume 11, pages 290-303, June, 2006.

[146] Nickel, J (2016). Mastering Identity and Access Management with Microsoft Azure. p. 84. ISBN 9781785887888. Retrieved July 20, 2018.

[147] Nipkow T, Paulso L C and Wenzel M. Isabelle/HOL — A Proof Assistant for Higher-Order Logic. LNCS 2283, Springer, 2002

[148] Norris, J R. (1998). Markov chains. Cambridge University Press. Retrieved 2016-03-04.

[149] Nudd G R, Kerbyson D J, Papaefstathiou E, Perry S C, Harper J S, Wilcox D V. PACE—A toolset for the performance prediction of parallel and distributed systems. The International Journal of High Performance Computing Applications, Volume 14, No. 3, Fall 2000, pp. 228-251

[150] Ölveczky P C, Meseguer J. Semantics and pragmatics of Real-Time Maude. *Higher-Order and Symbolic Computation*, 20(1-2):161–196 (2007).

[151] OMG. Unified Modeling Language: Superstructure, Version 2.3, 2010

[152] OMG. OMG Marte Web site, http://www.omgmarte.org/.

[153] OMG Unified Modeling Language Revision Task Force. "OMG Unified Modeling Language Specification". Version 1.5, March 2003. link:http://www.omg.org/technology/documents/formal/uml.htm

[154] OSGi Alliance. OSGi Service Platform Core Specification, Release 5.0, June 2012. http://www.osgi.org/Specifications/

[155] Padilla, Javier F, Frederic W, and Johann B. Towards a Model@Runtime Middleware for Cyber Physical Systems. In *Proceedings of the 9th Workshop on Middleware for Next Generation Internet Computing*, ACM, 2014. pages 6:1–6:6.

[156] Pastor O, España S, Panach J I, Aquino N, España S, Panach J Ignacio P, Aquino N. Model-Driven Development. Journal of Informatik-Spektrum Volume 31, Issue 5 , pp 394-407. 2008-10.

[157] Perera C, Liu C H, Jayawardena S and Chen M. A Survey on Internet of Things From Industrial Market Perspective. In the Journal of IEEE Access 2014, volume 2, pages 1660-1679. DOI: 10.1109/ACCESS.2015.2389854

[158] Pinet F, Kang M A and Vigier F. (2005). Spatial constraint modelling with a gis extension of uml and ocl: Application to agricultural information systems. In Wiil, U., editor, *Metainformatics*, volume 3511 of *Lecture Notes in Computer Science*, pages 160–178. Springer Berlin Heidelberg.

[159] Pnueli A. The Temporal Logic of Programs. In Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, pages 46-57, 1977.

[160] Quaas K. MTL-Model Checking of One-Clock Parametric Timed Automata is Undecidable. Proceedings 1st International Workshop on Synthesis of Continuous Parameters, SynCoP 2014, Grenoble, France, 6th April 2014.

[161] Radhakisan B and Gill H, "Cyber-physical systems", The Impact of Control Technology, IEEE, pp. 161-166, 2011.

[162] Rafe V and Rahmani A. Towards automated software model checking using graph transformation systems and bogor. *Journal of Zhejiang University - Science A*, 10:1093–1105. 10.1631/jzus.A0820415 (2009).

[163] Rajeev A, Courcoubetis C, Dill D, Halbwachs N, Howard W T. An implementation of three algorithms for timing verification based on automata emptiness. In Proc. 13th IEEE Real-Time Systems Symposium (RTSS'92), pages 157–166. IEEE Computer Society Press, 1992.

[164] Rajeev A, Courcoubetis C, Henzinger T A. The observational power of clocks. In Proc. 5th International Conference on Concurrency Theory (CONCUR'94), volume 836 of Lecture Notes in Computer Science, pages 162–177. Springer, 1994.

[165] Rajeev A, Fix L and Henzinger T A. A determinizable class of timed automata. In Proc. 6th International Conference on Computer Aided Verification (CAV'94), volume 818 of Lecture Notes in Computer Science, pages 1–13. Springer, 1994.

[166] Rana, R., Staron, M., Hansson, J., Nilsson, M., Meding, W. A framework for adoption of machine learning in industry for software defect prediction. In: 9th International Conference on Software Engineering and Applications, pp. 383–392. IEEE (2014)

[167] Reed G M and Roscoe A W. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58(1-3):249–261. 1988.

[168] Sabaliauskaite G and Mathur A P. Aligning Cyber-Physical System Safety and Security, In Complex Systems Design & Management Asia: Designing Smart Cities: Proceedings of the First Asia - Pacific Conference on Complex Systems Design & Management, CSD& M Asia 2014. Springer International Publishing, 2015. pages 41-53.

[169] Schmittner C, Ma ZD and Gruber T. Combining Safety and Security Engineering for Trustworthy Cyber-Physical Systems. the European Research Consortium for Informatics and Mathematics (ERCIM) news online edition. link: http://ercim-news.ercim.eu/en102/special/combining-safety-and-security-engineering-for-trustworthy-cyber-physical-systems

[170] Selic B. (1998). Using UML for modeling complex real-time systems. In Mueller, F. and Bestavros, A., editors, *Languages, Compilers, and Tools for Embedded Systems*, volume 1474 of *Lecture Notes in Computer Science*, pages 250–260. Springer Berlin Heidelberg.

[171] Sendall S and Kozaczynski W. Model Transformation: The Heart and Soul of Model-Driven Software Development. IEEE Softw. 20, 5 (September 2003), 42-45.

[172] SOA for the Business Developer: Concepts, BPEL, and SCA. ISBN 978-1-58347-065-7

[173] Spichkova M. Stream Processing Components: Isabelle/HOL Formalisation and Case Studies. In Archive of Formal Proofs, ISSN 2150-914x, 2013.

[174] Spichkova M, Blech J O, Herrmann P and Schmidt H. 11th Workshop on Model Driven Engineering, Verification and Validation (MoDeVVa 2014 @ MODELS). 2014.

[175] Shu G and Hsu Y and Lee D, Realtime Garbage Collection in the JamaicaVM 3.0, in Proceedings of Formal Techniques for Networked and Distributed Systems – FORTE 2008, 28th IFIP WG 6.1 International Conference Tokyo, Japan, June 10-13, 2008.

[176] Siebert F. Realtime Garbage Collection in the JamaicaVM 3.0, in Proceedings of the 5th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES '07), pages 94-103, Vienna, Austria, 2007.

[177] Slåtten V. Model Checking Collaborative Service Specifications in TLA with TLC. Project Thesis (2007) Norwegian University of Science and Technology

[178] Slåtten V. Automatic Detection and Correction of Flaws in Service Specifications. Master's thesis, Norwegian University of Science and Technology, June 2008.

[179] Slåtten V. *Towards Model-Driven Engineering of Reliable Systems - Developing Fault-Tolerant Systems using Scalable Verification.* PhD thesis, Norwegian University of Science and Technology (2014).

[180] Slåtten V, Herrmann P. (2011). Contracts for multi-instance uml activities. In Bruni, R. and Dingel, J., editors, *Formal Techniques for Distributed Systems*, volume 6722 of *Lecture Notes in Computer Science*, pages 304–318. Springer Berlin Heidelberg.

[181] Slåtten V, Herrmann P, Kraemer F A. Chapter 4 — Model-Driven Engineering of Reliable Fault-Tolerant Systems—A State-of-the-Art Survey. In Advances in Computers, Elsevier. Volume 91, pages 119-205.

[182] Slåtten V, Kraemer F A, Herrmann P. Towards Automatic Generation of Formal Specifications to Validate and Verify Reliable Distributed Systems: A Method Exemplified by an Industrial Case Study. In *Proceedings of the 10th ACM International Conference on Generative Programming and Component Engineering*, *ACM*. pages 147–156.

[183] Song S, Zhang J, Liu Y, Auguston M, Sun J, Dong J and Chen T.

Formalizing and verifying stochastic system architectures using monterey phoenix. *Software and Systems Modeling*, pages 1–19 (2014).

[184] SSO and LDAP Authentication. Authenticationworld.com. Archived from the original on 2014-05-23. Retrieved 2014-05-23.

[185] Straeten R V D, Mens T, Baelen S V. Challenges in Model-Driven Software Engineering. In *Models in Software Engineering*, vol. 5421, Lecture Notes in Computer Science, pages 35–47. 2009.

[186] Støyle A K. Service Engineering Environment for AMIGOS *Master's thesis*, Norwegian University of Science and Technology, 2004.

[187] Sun J, Liu Y, Dong J S, Liu Y, Shi L and André É. Network simulations with OPNET. In Journal of ACM Transition. Software. Engineering. Methodology, vol. 22, ACM, pages 3:1–3:29, 2013.

[188] Szanto Z, Marton L, Haller P and GyorgyS. Performance Analysis of WLAN based Mobile Robot Teleoperation. In *IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 299–305, IEEE Computer, 2013.

[189] Taentzer, G. AGG: A graph transformation environment for modeling and validation of software. In Pfaltz, J L, Nagl M and Böhlen, B., editors, *Applications of Graph Transformations with Industrial Relevance*, volume 3062 of *Lecture Notes in Computer Science*, pages 446–453. Springer Berlin / Heidelberg (2004).

[190] Thomas W, "Automata on infinite objects". In Van Leeuwen. Handbook of Theoretical Computer Science. Elsevier. pp. 133-164.

[191] Trivedi K S, Haverkort B R, Rindos A, Mainkar V. Techniques and tools for reliability and performance evaluation: Problems and perspectives. Proceeding of 7th International Conference on Computer Performance Evaluation Modelling Techniques and Tools (TOOLS), Vienna, Austria, May 3–6, 1994

[192] Universität Hamburg. Petri Nets Tools Database Quick Overview, https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html.

[193] UP4ALL. UPPAAL language reference, Web link:http://www.uppaal.com/index.php?sida=217&rubrik=101

[194] Varró D, Asztalos M, Bisztray D, Boronat A, Dang D, GeißR, Greenyer J, Pieter V, Kniemeyer O, Narayanan A, Rencis E, Weinell E. Transformation of UML Models to CSP: A Case Study for Graph Transformation Tools. in: Applications of Graph Transformations with Industrial Relevance, pp. 540–565. Springer-Verlag (2011).

[195] Wang, F. Formal Verification of Timed Systems: A Survey and Perspective. *Proceedings of the IEEE*, Vol. 92, No. 8, August 2004. pp. 1283-1305.

[196] Wang J. January 1, 2012. Handbook of Finite State Based Models and Applications. published by Chapman & Hall/CRC.

[197] Wang, S Y and Lin C C. Modeling and Verifying Hierarchical Real-time Systems Using Stateful Timed CSP In *68th IEEE Vehicular Technology Conference*, pages 1–2, IEEE Computer, 2008.

[198] Wang Y. CCS + time = an interleaving model for real time systems. *In Proceedings, Eighteenth International Colloquium on Automata, Languages and Programming, volume 510 of Lecture Notes in Computer Science.*, Springer-Verlag, 1991.

[199] Wilke T. Specifying timed state sequences in powerful decidable logics and timed automata. In Proc. 3rd International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94), volume 863 of Lecture Notes in Computer Science, pages 694–715. Springer, 1994.

[200] Yashkov, S. F.. Processor-sharing queues: Some progress in analysis. Queueing Systems, 2:1-17, 1987.

[201] Yu Y, Manolios P, Lamport L. Model Checking TLA+ Specifications. In Pierre, L., Kropf, T., eds.: Proc. 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'99). LNCS 1703, Springer-Verlag (1999) 54–66

[202] Zhang, D., Tsai, J.J. Machine Learning and Software Engineering. Software Quality Journal 11, 87–119 (2003). https://doi.org/10.1023/A:1023760326768

[203] Zhang J, Lin Y and Gray J. Generic and domain-specific model refactoring using a model transformation engine. In *Volume II of Research and Practice in Software Engineering*, pages 199–218. Springer (2005).

[204] Zhao J and Dang H. On checking parallel real-time systems for linear duration properties. In Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems: 5th International Symposium, FTRTFT'98 Lyngby, Denmark, pages 241–250, September 14–18, 1998 .

[205] Zhou C, Hoare C A R and Ravn A P. A Calculus of Durations, Information Processing Letters, 40(5):269–276, December 1991.

[206] Zhou C C. Duration calculus, a logical approach to real-time systems. *Lecture Notes in Computer Science*, 1548:1–7, 1999.

[207] Zurowska K. Domain specific analysis of statemachine models of reactive systems. In *Demos/Posters/StudentResearch@MoDELS'13*, pages 81–86 (2013).

II

INCLUDED PAPERS

PAPER A

Towards Choreography Model Transformation via Graph Transformation

Fenglin Han, Surya Bahadur Kathayat, Hien Le, Rolv Braek and Peter Herrmann

Proceedings of the 2nd IEEE International Conference on Software Engineering and Service Sciences (ICSESS 2011). Beijing, 15-17 July 2011.

TOWARDS CHOREOGRAPHY MODEL TRANSFORMATION VIA GRAPH TRANSFORMATION

Fenglin Han
Norwegian University of Science and Technology, Trondheim, Norway
sih@item.ntnu.no

Surya Bahadur Kathayat
Norwegian University of Science and Technology, Trondheim, Norway
surya@item.ntnu.no

Hien Le
Norwegian University of Science and Technology, Trondheim, Norway
hiennam@item.ntnu.no

Rolv Braek
Norwegian University of Science and Technology, Trondheim, Norway
rolv.braek@item.ntnu.no

Peter Herrmann
Norwegian University of Science and Technology, Trondheim, Norway
herrmann@item.ntnu.no

Abstract We present a Model-Driven method to develop collaborative systems. In our method, we use UML collaborations to capture the requirements and architecture of such a system. The system behavior is specified by two choreography models: an abstract flow-global and a more detailed flow-localized choreography. These choreography models are both described by UML activity diagrams. A graph-based transformation approach carrying out the transformation from the flow-global to the flow-local choreography is the core contribution of this paper. Our approach is illustrated using a case study of the European Rail Traffic Management System (ERTMS).

Keywords: Choreography Model, Model Transformation, Graph Transformation, Collaborative Service.

1 INTRODUCTION

Model-Driven Development (MDD) is an approach supporting the software development process by creating models on different levels of abstraction and platform independence. First, one develops more

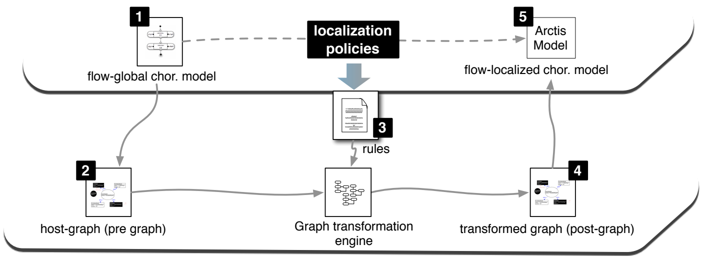


Figure A1. Overall Approach.

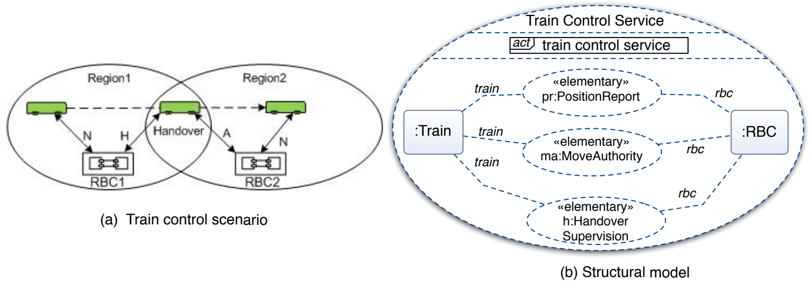

Region1
Region2
Handover
N
H
A
N
RBC1
RBC2
(a) Train control scenario
Train Control Service
act train control service
:Train
:RBC
train
train
train
rbc
rbc
rbc
«elementary» pr:PositionReport
«elementary» ma:MoveAuthority
«elementary» h:Handover Supervision
(b) Structural model

Figure A2. Structure Model of the Train Control System.

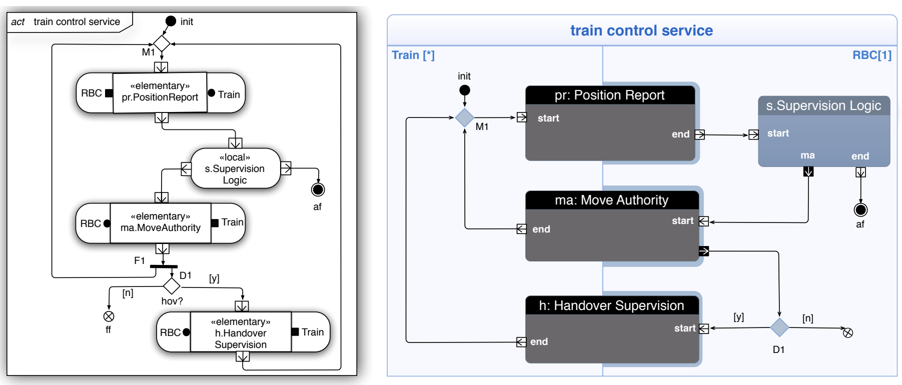
abstract models specifying the pure functionality of a particular solution or an application domain but hiding aspects of the later realization. These models can be transformed into models incorporating more implementation details. Based on the refined models, application code can be generated ranging from system skeletons to complete, deployable products for different platforms. MDD is considered effective when the transformation from the abstract to the detailed models can be done with a high degree of automation. This makes it easy to keep consistency between the two model levels. In addition, the developer can utilize the comprehensibility and the generality of the platform-independent more abstract model as well as the fine-grained semantics and the mature structuring mechanisms of the more detailed one.

In this paper we discuss a model-driven development method for distributed, reactive and collaborative services. A collaborative service is defined as a partial system functionality in which two or more components collaborate to achieve a common goal [4, 11]. UML 2.x collaborations are used to describe the structure of participants cooperating with each other, while UML activities specify the corresponding behavior. According to the objectives of MDD, we consider the behavior models (called choreography models in the following) at two levels of detail and use graph-based model transformation to derive detailed implementable models from more global abstract ones.

Figure A1 delineates the overall MDD approach:

- A flow-global choreography model (label 1 in Figure A1) seeks to specify the desired global behavior in terms as close to the problem domain as possible. It is intended to be understandable by end-users and experts of a specific domain. Thus, the flow-global choreography focuses on the global interaction while the details and resolutions of coordination problems that may occur at the level of a distributed realization are not modeled. We express flow-global choreographies by a special kind of UML 2.x activities [7].

- A flow-localized choreography (label 5 in Figure A1) is used to define global behavior in sufficient detail that coordination problems arising in a distributed system can be resolved. Further, the level of detail shall allow extensive analysis, synthesis of the behavior in the distributed system parts, and automatic generation of the application code. In particular, we apply the system engineering



(a) Flow-global choreography model

(b) Flow-localized choreography model

Figure A3. Flow-Global model and Flow-Localized model (derived as Arctis model).

approach SPACE [17] and the corresponding tool-set Arctis [2] that also uses UML 2.x activities to model behavior. The models created in Arctis can fully automatically transformed to Java code running on several platforms [11].

- Flow-global choreographies can be transformed to flow-localized choreographies using graph transformation techniques. The localization policies are implemented by applying graph transformation rules (label 3 in Figure A1) to a host-graph representing a flow-global choreography model (label 2) to derive a post-graph of a flow-localized choreography specification (label 4).

The focus of this paper is on the model transformation using a graph transformation engine. This provides for the following advantages: First, the flow-global models, which can be re-used in different system scenarios, are stored in domain-specific repositories [7]. In a similar way, we can predefine graph transformation rules for certain refinements which are stored in repositories as well and can be used for different kinds of transformations. Second, it is not necessary that the full flow-global model must already be available in order to be transformed to a flow-localized model. This is due to the possibility to transform partial flow-global choreography models.

We introduce the structure and choreography models by a train control system scenario in Section 2. A survey of the task for the model transformation is provided in Section 3. Model transformation using the graph transformation approach is presented in Section 4. Section 5 discusses the related work and is followed by concluding remarks in Section 5.

2 ARCHITECTURE AND CHOREOGRAPHY

In this section, we discuss the related models contributing to our MDD approach:

- The collaboration model that defines service participants as roles and sub-services as collaboration uses.

- The flow-global choreography model specifying the high-level global behavior including the ordering and causality among sub-services and service roles in a composite service.

- The flow-localized choreography model defining detailed behavior so that application code can be generated.

We present a summary of these models using an example of the European Rail Traffic Management System (ERTMS).

2.1 Collaboration

UML collaborations are used to specify the structure (i.e., roles and interactions) of distributed collaborative entities and collaboration uses representing sub-collaborations. Figure A2 (a) shows an example of a train control service which is described as following: A train must always be supervised by a radio block center (*RBC*). The *RBC*'s responsibility is to monitor and control all train movements in a particular region. Guided by its current *RBC*, the train keeps on moving and sends its position reports to the *RBC* and the *RBC* validates the received position information of the train. Moreover, the *RBC* issues successive movement authorities (*MA*) to the train, which specifies a safe distance the train may travel. In addition, the train may also travel across several regions covered by different *RBC*s. This means, the supervision of the train movement can be handled by more than one *RBC*. When the train crosses a border between two regions, the *RBC* of the current region will hand over the control to another one by executing a handover process. The train control service is modelled as a UML collaboration as shown in Figure A2 (b).

The *TrainControlService* system has two main participants,*Train* and *RBC*, represented as roles. In Figure A2 (b), the *Train* and the *RBC* participate in three collaborative sub-services: *PositionReport*, *MoveAuthority*, and *HandoverSupervision* which perform the following activities:

1 *PositionReport* reports the current train position to the RBC.

2 *MoveAuthority* sends the safe travel distance from the RBC to the train.

3 *HandoverSupervision* transfers the train control supervision to the new RBC if the train travels to a different region.

2.2 Flow-global choreography models

The flow-global choreography shown in Figure A3 (a) is defined by UML activities connecting actions (in particular, call behavior actions, i.e., actions including own behavioral models) by flows that are not assigned to any particular role in the collaboration. Actions may either specify the behavior of a collaboration or a local activity. Collaborative actions contain references to their participants in the form of roles. Further, we indicate initiating and terminating roles by dots respective squares. Note that the pins are not localized to the roles in this model type.

Flows may contain intermediate control nodes (e.g. start, stop, choice, merge, fork and join) defining the ordering and causality among the actions. Like the pins, the control nodes are not assigned to any particular component, too.

Thus, the flow-global choreography model abstracts from several design issues that need to be addressed when transforming it to a flow-localized choreography model. We believe that this is the right level of abstraction to discuss the intended behavior with end-users and other stake holders since global choreography models specify the global action order without describing detailed interactions as in interaction diagrams. They hide details needed in a distributed realization such as the location of decisions and other control nodes, and coordination details ensuring that the global ordering is satisfied. This pure focus on the functional behavior of a system allows to gain a better understanding of the system behavior and to find potential development errors early. Further, the constriction on functionality reduces the state space produced by model checkers verifying certain system properties which eases the use of these automatic analysis tools also for more complex systems. All-in-all, a flow-global choreography model is a useful first step in the formalization of the requirements of a system.

Figure A3 (a) depicts the flow-global choreography behavior of the train control service. A train on its journey reports its current position in intervals to the *RBC* which is responsible for the region the train operates in. This operation is specified by the collaboration *PositionReport*. Thereafter, the *RBC* validates the received position information of the train via the local activity *SupervisionLogic*. If the information about the location of the train is correct, the *RBC* issues successive movement authorities (MA) to the train which is modeled by the collaboration *MoveAuthority*. Finally, if the train crosses the border between two regions, the collaboration *HandoverSupervision* is invoked.

2.3 Flow-localized choreography models

As already mentioned, the flow-localized choreographies are modeled in Arctis [2], our tool-set to develop component based collaborative systems which specifies behavior using UML activities as well. Also here, actions representing the behavior of a collaboration or a local activity are connected by flows and intermediate control nodes. In contrast to the activities introduced above, however, all nodes and pins are each localized to a role specifying a participating distributed entity. The roles are represented in an activity by partitions. Flows that cross partition boundaries thereby imply communication and transfer delays.

This location information allows to analyze the flow-localized choreography for realization problems like for instance mixed initiatives in which two different physical components concurrently initiate co-operations which due to the transmission lag are not properly detected and may lead to unpredicted erroneous behavior.

In Arctis, the activities are provided with a formal semantics [20] which allows for the application of model checkers to detect design errors [11]. Further, application code for different platforms such as Standard Java, Android and Sun Spots can be automatically generated from the Arctis building blocks. In the train control scenario, this is the code for the train and RBC components.

A screen shot of the Arctis model of the train control system is shown in Figure A3 (b). The collaborative and the local actions are represented as Arctis building blocks (the dark gray respectively blue boxes with pins). Control nodes are localized to components represented by the Activity partitions. The extra gray border of the collaborative actions (*pr*, *ma* and *h*) at the *RBC* part denotes that the service roles bond to this part are multiple-session. This specifies that an *RBC* may cooperate with several trains at the same time.

3 FLOW LOCALIZATION

Given the two choreography models, we outline the overall transformation process. First, we define the causal relationship between sequential collaborative activities. Second, we introduce a localization policy based on the causal properties.

3.1 Causality relationship

In order to localize the flows and control nodes between actions in a flow-global choreography, we first need to classify the causality among actions that follows directly from the flow-global choreography. As described in [4, 7], the following causal relationships between any two sequential connected actions C_1 and C_2 can be:

- *Strong flows:* The terminating role of C_1 and the initiating role of C_2 belong to the same system component. In this case, the flow between C_1 and C_2 can be executed locally by this component.

- *Non-causal flows:* The initiating role of C_2 belongs to a component that does not participate in C_1. This means that local ordering between actions of C_1 and C_2 cannot be achieved by a local flow. Here, we need communication between different components.

- *Weak flows:* The initiating role of C_2 belongs to the same component as a non-terminating role in C_1. Here, the non-terminating role of C_1 and the initiating role of C_2 can be ordered by a local flow, but one has to be aware that other roles in C_1 may not be finished when C_2 starts, and both collaborations run in parallel for a while.

3.2 Localization policy

In the simplest case where there are no intermediate control nodes between actions (i.e., direct-flows), a global flow from C_1 to C_2 is localized as follows:

- Localize *strong* flows to the role that initiates C_2 and terminates C_1.

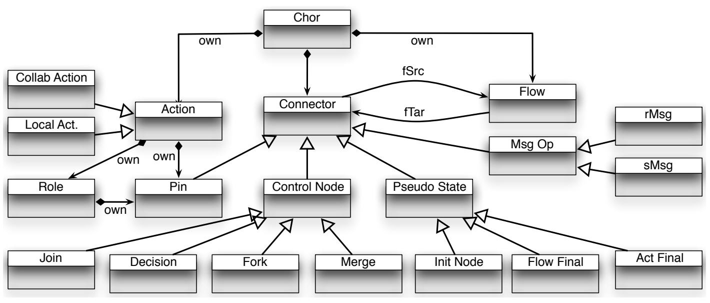


Figure A4. Meta-model for the choreography graph models.

- *Non-causal* flows can only be maintained using send and receive events realizing communication between the role terminating C_1 and the one initiating C_2. In Arctis, this is modeled by flows passing partition borders. We call this procedure *enforced* strong sequencing [7].
- In the case of *weak* flows we have two alternatives. We can either use enforced strong sequencing as well or add an extra streaming pin to C_1 which, however, changes the internal behavior of this action slightly. For this modification we prepared a set of transformation rules for weaving extra behavior into building blocks without effecting the original functional design. This will not be addressed in this paper due to the space limitation.

If there are one or more intermediate control nodes between actions C_1 and C_2, one must consider all possible flow-paths passing through them. This means that each control node can be part of several paths. We use the following notation to represent the paths and path property:

$$Path ::= (sourceNode) \overset{causality}{\rightarrow} (targetNode)$$

where $sourceNode$ and $targetNode$ are either pseudo nodes (such as an initial node or an activity final node which are represented by $\star$) or collaboration role identifiers (represented by collaborationId.roleId). The arrows show the flow direction between those collaborative activities and contains a mark for the causality relation on top. In the flow-global choreography model in Figure A3 (a), there are in total eight paths:

- $P_0 : \star \overset{na}{\rightarrow} pr.Train;$
- $P_1 : pr.RBC \overset{strong}{\rightarrow} s.RBC;$
- $P_2 : s.RBC \overset{strong}{\rightarrow} ma.RBC;$
- $P_3 : ma.Train \overset{strong}{\rightarrow} pr.Train;$
- $P_4 : ma.Train \overset{na}{\rightarrow} \star;$
- $P_5 : ma.Train \overset{weak}{\rightarrow} h.RBC;$
- $P_6 : h.Train \overset{strong}{\rightarrow} pr.Train;$
- $P_7 : s.RBC \overset{na}{\rightarrow} \star;$

The abbreviation *na* means that there is no causality relationship available since the start or end of the flow is a pseudo node. P_0, P_4 and P_7 are such dangling paths in which the pseudo state and the control nodes along the path will be localized to the activity linked to the path. For instance, since the initial node occurs only in P_0, it will be localized to the role *Train*. In order to localize the remaining control nodes M_1, D_1 and F_1 and paths P_1, P_2, P_3, P_5 and P_6, we need to consider the path properties that each control node is involved in:

- M_1: $P_6(strong)$, $P_3(strong)$;
- F_1: $P_3(strong)$, $P_5(weak)$;
- D_1: $P_5(weak)$.

M_1 fulfills *strong* causality for both involved paths and is therefore localized to *Train*. In contrast, F_1 and D_1 contain *weak causal* paths such that we need to find suitable breaking points along the involved paths, i.e., P_5 containing both F_1 and D_1. To achieve that, we assign a breaking priority level to all the nodes on the path. The priority level is defined by the combination of causality properties in Table A1. Here, the columns and rows represent the causality property of a path through a node. Altogether, we define seven breaking priority levels of which 1 refers to the lowest and 7 to the highest priority.

Table A1. Localization priority order and policy matrix for control node.

property	strong	weak	non-causal	all
strong	1	2	3	-
weak	-	5	6	-
non-causal	-	-	7	-
all	-	-	-	4

In the *TrainControlSystem* specification, D_1 is involved in $P_5(weak)$ and has priority level 5 according to Table A1. Similarly, F_1 is involved in $P_3(strong)$ and $P_5(weak)$ and has priority level 2. According to our policy, D_1 is selected as breaking node as it has the higher breaking priority level. In this case D_1 is broken at the incoming edge, i.e., if we decide to resolve the weak flow by enforced strong sequencing, the communication is provided at the edge between F_1 and D_1. However, in the succeeding refinement steps, we decide to add a streaming pin to $ma.MovementAuthority$ on the side of the *RBC* from which the edge to D_1 begins. Then, the fork F_1 has only one outgoing edge remaining and will therefore be removed. After finishing the various transformation steps, the constituted flow-localized graph model is transferred to the Arctis specification shown in Figure A3 (b).

4 Graph-based Model Transformation

This section discusses in which way graph transformation techniques can be used for model transformation from the flow-global choreography model to the flow-localized model. In the following, we describe the definition of the graph models, the graph transformation rules and implementation aspects.

4.1 Graph model definition

The meta-model for the choreography graph model (also called *type graph* in the following) is shown in Figure A4 in terms of UML class diagrams. A choreography type graph depicted by the class *Chor* has three main entities: *Actions*, *Connectors*, and *Flows*. An action can be either a *local activity* performed by only one specific role or a *collaborative activity* carried out by the cooperation of at least two roles. Connectors represent the mechanism to connect Actions, i.e., how collaborations connect to other collaborations or local activities. There are three types of connectors: *Pin*, *Control Node* and *Pseudo State*. Pins are connection points which are associated with either Roles (in the flow-localized form) or Actions (in the flow-global form). Control Nodes include join, fork, merge and decision nodes. Pseudo Nodes include initial nodes, activity final nodes and flow final nodes. Connectors can also be message operators (*MSg Op*), i.e., send message actions (*sMsg*) or receive message actions (*rMsg*).

Moreover, there are three types of graph edges: *own* specifies that one node is owned by another which is described by aggregations in the meta-model. Further, *fSrc* models the source node of the flow while *fTar* specifies the flow target.

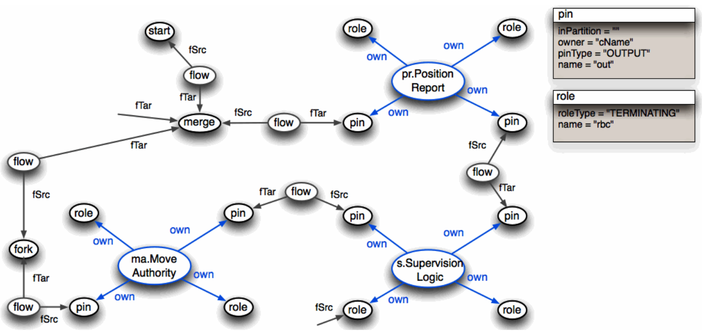


Figure A5. Graph model of the part of choreography graph model of Train control scenario

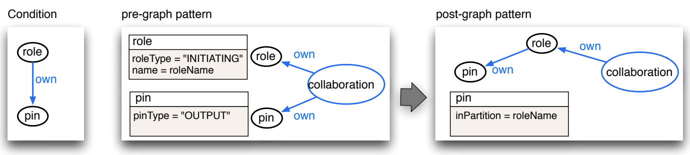


Figure A6. Pin localization rule graphs.

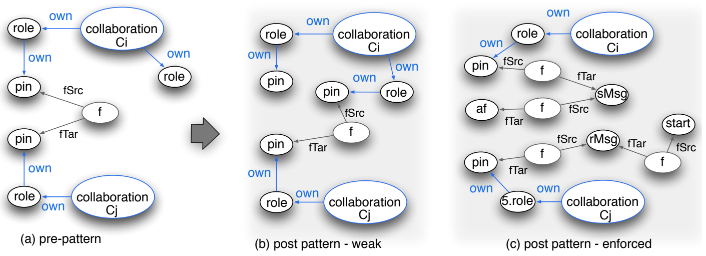


Figure A7. Direct flow localization rule.

Based on this type graph, two kinds of graph models can be defined corresponding to the flow-global and flow-localized choreography models. Figure A5 illustrates a flow-global graph representation of the train control system in Figure A3 (a) without the Handover activity. Note that there are two dangling edges in the graph model: *fTar* to a merge node and *fSrc* to a role node. These edges will eventually be connected to the Handover activity.

4.2 Graph models of the transformation rules

Transformation rules can be visualized and are mainly composed of two graph parts: the pre-pattern subgraph, expressing what to replace, and the post-pattern subgraph describing the replacement. A transformation condition can also be used to define conditions or constraints describing how and under which conditions the graph production can be applied (such as a negative application condition NAC introduced in [Tae04]). In the following, we introduce the graph model transformation rules and the policies through some representative rules: the pin location rule and the direct flow localization rule.

Graph models of the pin localization rule

Figure A6 depicts the graph transformation rule for pin allocation. Pins in the pre-pattern of the graph model are owned by collaboration nodes as shown on the left side of Figure A6. They are localized, i.e., connected to roles in the post-pattern of the graph as depicted on the center and right sides of Figure A6. Note that attributes and their values attached to the graph nodes can be used, checked, or modified. For example, the attributes *roleType* and *pinType* are checked in the nodes *role* and *pin* in the pre-pattern of the graph model. Similarly, the attribute *inPartition* of the pin node is modified (or assigned a value) in the post-pattern of the graph.

Graph models of direct flow localization rules

Figure A7 shows the pre-pattern and post pattern of the rules which localize the direct flow having *weak* causality and *non-causal* causality properties. Figure A7 (a) depicts the pre-pattern graph model of the direct flow in which pin allocation rules have been performed. If the flow is *weak* causal, the corresponding post-pattern is shown in Figure A7 (b). In this case, a new output streaming pin is added to a collaborative activity *Ci*. An example of this type is P_5 in the case study. In the case that a direct flow-path is *non-causal*, the flow-path is resolved using send and receive message nodes as shown by the post-pattern graph model in Figure A7 (c).

4.3 Implementation

An Eclipse plug-in has been developed to create graph models of flow-global choreography models and to import the post-graph (as a result of transformation) into Arctis. As graph transformation engine, we use the Attributed Graph Grammar System (AGG) [Tae04]. AGG offers high flexibility in creating the visual definition of graph models. Further, it provides Java APIs facilitating its integration to the Arctis tool which is also Java-based. AGG has also a facility to enable the verification and correctness of models during transformation.

The AGG graph transformation engine takes the graph model of a flow-global choreography as well as the rules introduced in Section 4.2 as inputs and produces the post-graph model which corresponds to a flow-localized choreography.

5 RELATED WORK

A comparison of approaches and tools that use graph transformation techniques for model transformations is provided in [2]. In [3], Kerkouchea et al. propose an approach for transforming UML state-chart and collaboration diagrams to Colored Petri nets using graph transformation techniques. In contrast, the authors of [1, 2] suggest to map activity diagrams into communicating sequential processes (CSP).

Unlike these approaches, both our abstract and detailed models are based on UML activity diagrams. Moreover, our approach envisages collaborative building blocks encapsulating the interaction between different components. UML activities can be defined hierarchically by means of call behavior actions. We use AGG as our graph transformation engine and our post graphs can be directly imported to the Arctis tool for further analysis, synthesis and code generation.

In contrast to [6, 15], we currently miss the formal proof that our graph-based transformation is correctness-preserving. Due to the formal semantics of Arctis [20], however, which can also be used for the flow-global choreography models, the correctness verification can be provided as temporal logic-based refinement proofs [13] which is intended to be done for the close future.

Ideally, one can choose any graph transformation tool to validate or implement our approach. Some candidates other than AGG are ATOM3 [12], VTMS [14] and C-SAW [17]. As mentioned above, we chose AGG due to its high flexibility and its Java-compliance.

6 CONCLUDING REMARKS

In this paper, we presented a Model-Driven Development approach to support the engineering process of distributed collaborative services. The global behavior and distributed realization are captured by two different types of choreography models: flow-global and flow-localized, which are specified using UML activity diagrams. The transformation between these two choreography models are performed with the support of graph transformation techniques. The approach is used within the EU-funded project CESAR for the cost-effective development of safety-relevant embedded systems [5]. As future work, we plan to refine the model transformation policies and test them with larger and more complex system models. As mentioned above, we will further prove the correctness of the graph transformation rules formally.

Graph transformation is also useful for automatic collaborative building block refinement. We currently developed a set of graph transformation rules to detect and correct mixed initiatives in distributed systems. As mentioned earlier, in this often occurring but not corrected class of errors, two system components may concurrently start cooperations which might lead to unpredictable system errors. Altogether, we are convinced that graph transformation is a highly promising extension to MDD which, essentially, consists of the stepwise refinement of graphical notations. Here, the flexibility of this rule-based approach may facilitate the refinement steps significantly.

Bibliography

[1] Elmansouri, R., Hamrouche,H., Chaoui, A.: From UML Activity Diagrams to communication sequential processes (CSP) Expressions: A Graph Transformation Approach using Atom3 Tool. International Journal of Computer Science, 8(2), 2011.

[2] Varró, D., Asztalos, M., Bisztray, D., Boronat, A., Dang, D., Geiß, R., Greenyer, J., Pieter, V., Kniemeyer, O., Narayanan, A., Rencis, E., Weinell, E.: Transformation of UML Models to CSP: A Case Study for Graph Transformation Tools. in: Applications of Graph Transformations with Industrial Relevance, pp. 540–565. Springer-Verlag (2011).

[3] Kerkouchea, E., Chaouib, A., Bourennanec, E., Labbanic, O.: A UML and Colored Petri Nets Integrated Modeling and Analysis Approach using Graph Transformation. In: Journal of Object Technology, vol. 9, no. 4, pp. 25D43. JOT (2010)

[4] Castejon, H., Braek, R., and von Bochmann, G. (2007). Realizability of collaboration-based service specifications. In *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, pages 73 –80.

[5] CESAR (2010). http://www.cesarproject.eu/ Accessed Jan 2011.

[6] Jayaraman, P., Whittle, J., Elkhodary, A., and Gomaa, H. (2007). Model composition in product lines and feature interaction detection using critical pair analysis. In Engels, G., Opdyke, B., Schmidt, D., and Weil, F., editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *Lecture Notes in Computer Science*, pages 151–165. Springer Berlin / Heidelberg.

[7] Kathayat, S. B. and Bræk, R. (2010). From flow-global choreography to component types. In *System Analysis and Modeling (SAM)*, volume 6598 of *Lecture Notes in Computer Science*. Springer - Verlag.

[8] Kraemer, F. A. (2008). *Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks*. PhD thesis, Norwegian University of Science and Technology.

[9] Kraemer, F. A., Bræk, R., and Herrmann, P. (2009) Compositional Service Engineering with Arctis. *Telektronikk*, 105(2009)1.

[10] Kraemer, F. A., and Herrmann, P. (2010) Reactive Semantics for Distributed UML Activities. In *Formal Techniques for Distributed Systems*, volume 6117 of *LNCS*, pages 17–31, 2010.

[11] Kraemer, F. A., Slåtten, V., and Herrmann, P. (2009). Tool support for the rapid composition, analysis and implementation of reactive services. *Journal of Systems and Software*, 82(12):2068 – 2080.

[12] Lara, J. and Vangheluwe, H. (2002). Atom3: A tool for multi-formalism and meta-modelling. In Kutsche, R.-D. and Weber, H., editors, *Fundamental Approaches to Software Engineering*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer Berlin / Heidelberg.

[13] Lamport, L. (1994). The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923.

[14] Lengyel, L., Levendovszky, T., Mezei, G., and Charaf, H. (2006). Model transformation with a visual control flow language. *International Journal of Computer Science (IJCS)*, 1(1):45–53.

[15] Rafe, V. and Rahmani, A. (2009). Towards automated software model checking using graph transformation systems and bogor. *Journal of Zhejiang University - Science A*, 10:1093–1105. 10.1631/jzus.A0820415.

[16] Taentzer, G. (2004). AGG: A graph transformation environment for modeling and validation of software. In Pfaltz, J. L., Nagl, M., and Böhlen, B., editors, *Applications of Graph Transformations with Industrial Relevance*, volume 3062 of *Lecture Notes in Computer Science*, pages 446–453. Springer Berlin / Heidelberg.

[17] Zhang, J., Lin, Y., and Gray, J. (2005). Generic and domain-specific model refactoring using a model transformation engine. In *Volume II of Research and Practice in Software Engineering*, pages 199–218. Springer.

PAPER B

Remedy of mixed initiative conflicts in model-based system engineering

Fenglin Han and Peter Herrmann

Proceedings of the 11th International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2012), volume 47 of of the Electronic Communications of the EASST, Tallinn, 2012.

In this paper we used a button game scenario to illustrate the overall approach, which was originally described in Vidar Vidar Slåtten's master thesis.

REMEDY OF MIXED INITIATIVE CONFLICTS IN MODEL-BASED SYSTEM ENGINEERING

Fenglin Han
Norwegian University of Science and Technology,
Trondheim, Norway
sih@item.ntnu.no

Peter Herrmann
Norwegian University of Science and Technology,
Trondheim, Norway
herrmann@item.ntnu.no

Abstract SPACE is a technique for model-driven engineering of reactive distributed systems. One of the strengths of its tool-set Arctis is that the system engineer can formally analyze the models for design errors such that these can be corrected early in the development process. In this paper, we go a step further and introduce a technique that refines the fault detection and, in addition, offers a highly automatic mechanism to remedy the errors. For that, we combine model checking, the already existing analysis method of Arctis, with graph transformation. Using graph rewriting rules, we can analyze the state space graph of a system for the exact reason of an error as well as remove the erroneous parts of a model by changing the model description. We exemplify the approach by envisaging the detection and remedy of mixed initiatives, a quite common cause for faulty behavior in event-driven systems that often is overlooked in system development.

Keywords: graph transformation, model driven engineering,mixed initiative.

1 Introduction

New application domains like sensor networks, smart grids, and machine to machine cooperation call for novel networked services and applications. To engineer these often reactive and embedded distributed systems, we provide the development method SPACE and its tool-set Arctis [22, 18]. System behavior is modeled by UML activities [4] that use a token semantics close to Petri nets. The activities have been provided with a new reactive formal semantics [5] that enables to analyze the models formally [22] and to create code automatically [17]. The technique is scalable since we can enclose partial behavior into UML call behavior actions that we call *blocks*. On the one hand, a block embraces an activity and, on the other, it can be used as an element in another one. So, it links both activities together. The static role-binding of the blocks is modeled by UML collaborations while we use External State Machines (ESMs, [18]) to define the interface behavior of a block. Furthermore, the block structure enables a high degree of reuse of system models which is in average 70% in our models [18].

Arctis enables the formal analysis of desirable system properties (e.g., verifying that the activity embraced in a block fulfills its ESM) by model checking [22]. Since model checking can be executed in a non-interactive way and the traces towards detected errors are animated on the UML activities, the Arctis user does not need a deeper understanding of the formalism used. The state explosion problem of model checking is mitigated by compositional verification since the ESMs allow to prove every activity in the system model separately [22].

Up to now, the Arctis analysis has not supported automatic remedy of detected errors which has to be provided manually by the system engineer. This paper describes a solution to support the Arctis user further. In particular, we introduce a way to analyze the state graph of an erroneous model and to correct the specification with a high degree of automation. For that, we apply graph transformation that already proved helpful for the model transformation of flow-global choreographies, a more abstract way to specify reactive systems, to Arctis models [HKL+11]. Graph transformation is suitable since UML activities, collaborations and state machines all are graphical description techniques which can be directly accessed by graph rewriting rules.

For the error detection, we align graph transformation with model checking in Arctis. In particular, we analyze the state space of an activity generated by the model checker to find out if the detected errors belong to a particular class. Thereafter, we use rules to correct the erroneous part of the activity. We illustrate our approach with a mechanism for the detection and remedy of mixed initiatives between two parties.

2 Mixed Initiatives

Mixed initiatives [GY84, BH93, SDW08] are a special form of race conditions that is often overseen when developing reactive distributed systems. A mixed initiative conflict may occur whenever two (or more) distributed components can trigger an interaction with each other at the same time (see [Flo03]). Due to the asynchronous communication between the two parties, both can start own initiatives before being notified that also the partner triggered one. Of course, this behavior can be mitigated but that often demands for a complex new functionality. While mixed initiatives in practice are often overlooked, the Arctis model checker detects that a system contains faulty behavior (see [9]). By the methods introduced in this paper, we can find out if a mixed initiative is indeed the reason of an error. Further, we introduce a graph-based approach to correct mixed initiatives between two entities. The only condition for the remedy mechanism is that the system behavior contains an interaction between the entities before the mixed initiative can take place.

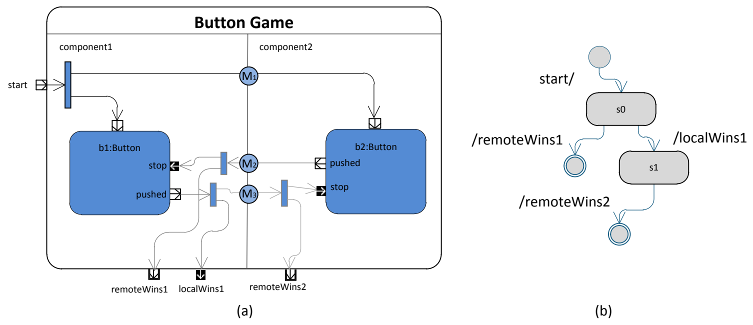


Figure B1. Arctis building block *Button Game* and its ESM

2.1 Arctis

The Arctis block *Button Game* depicted in Figure B1 (a) describes a simple game with two players which is won by the one who manages first to push a button. The players are represented by two technical components, e.g., two Android devices. As mentioned in the introduction, we model behavior by UML activities which are based on token flow semantics (see [5]). The activity shown in Figure B1 (a) is collaborative since it models the combined interaction of the two components *component 1* and *component 2* participating in the button game. To distinguish in which component a certain behavioral step takes place, the activity comprises two partitions marked by the component names.

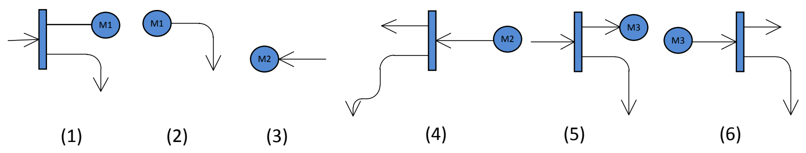


Figure B2. Activity steps of *Button Game*

A UML activity [4] is a directed graph and, due to the token flow semantics, behavior is modeled as tokens walking across the nodes of the graph following its edges. The edges may either stay within a partition, specifying local behavior of the corresponding component, or cross the partition borders (e.g., M_1, M_2, M_3). In the latter case, they model asynchronous interaction between two components. Activities offer a set of general node types enabling to start, stop, or interrupt token flows as well as nodes for routing and handling parallel flows respective for the execution of certain operations. An example are forks of which four copies are used in Figure B1 (a). They are expressed by bold bars in right angle to the linked edges. A fork contains one incoming edge and at least two outgoing edges and models that an incoming token is duplicated and a copy is sent via each of the downstream edges. Thus, forks enable to specify parallel flows.

Another node type used in the activity *Button Game* are call behavior actions describing the Arctis building blocks. A block represents an own activity that is linked with the one including it by means of pins[1] which are depicted as small rectangles on the edge of a block respective an activity and filled with in- or outgoing arrows. The interface behavior of a block is specified by External State Machines (ESM, [18]) that are simple UML state machines describing in which order flows may pass the various pins.

Figure B1 (a) includes the two blocks *b1* and *b2* of the type *Button* which are taken from an Arctis library for Android devices (see [Kra11]). This block type describes the logic when pushing a certain button of an Android device which initially is inactive. The button is armed by sending a token flow through its pin *start* on the top of the block. Thereafter, pushing the button leads to a flow via the pin *pushed* which, in addition, terminates and disarms the block. Further, the block can be disarmed from its environment by a token passing pin *stop*, too.

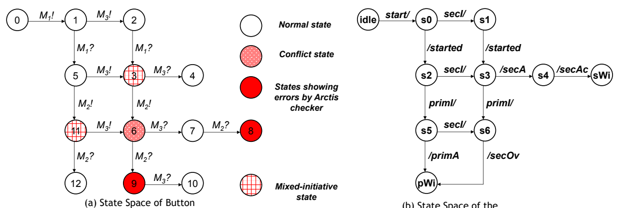
The Arctis semantics [5] defines so-called activity steps describing the sub-graph passed by a token in an atomic transition. In short, a token may rest only on nodes or edges describing places where it has to wait for a stimulus. An example are the crossing edges. To model the asynchronous communications between the partitions, a token passing a crossing edge has to wait on it until it is passed on in a new activity step. After being triggered by an internal or external event, e.g., the reception of the transmitted data, tokens pass all nodes and edges of the activity step in run-to-completion fashion until they reach nodes and edges on which they have to wait for new triggers.

Figure B2 shows the six activity steps of the UML activity in Figure B1 (a). Activity step (1) describes the start of the game. It is triggered by a token passing the parameter node *start* at the edge of the block *Button Game* which is duplicated at the fork node and copies are sent to both block *b1* and to the crossing edge M_1. Activity step (2) forwards the token to block *b2* such that both buttons are armed. Pushing a button leads to the activity steps (3) and (4) respective (5) and (6) which disarm the other button and notify the environment of the button game about the winner via the parameter nodes *localWins* and *remoteWins*.[2]

Figure B1 (b) shows the External State Machine (ESM, [18]) of the block *Button Game*. The block is started by a token passing pin *start*. Thereafter, it terminates either by a token arriving at *remoteWins1* or

[1]Formally, UML distinguishes between parameter nodes laying on the outer edge of an activity (e.g., *remoteWins1*) and pins on the edge of a block included in an activity (e.g., *stop*).

[2]For simplicity, we only notify *component 1*, that takes the role of the game manager, fully about the result while *component 2* is just informed if it lost.



(a) State Space of Button Game Building block before weaving

(b) State Space of the Button Game after adding block *Mixed Initiative 2*

Figure B3. State spaces of the original and the modified block *Button Game*

by one at *localWins1* followed by another one at *remoteWins2*. In the transition markings, the "/" behind a pin designator refers to tokens heading towards the block while positioning "/" in front refers to tokens coming from the block and going towards its environment.

2.2 A Mixed Initiative Error

It is easy to see that the system renders unexpected behavior if both buttons are pushed at the same time. Then due to the asynchronous communication between the components, the buttons will be terminated too late and tokens leave the pins *pushed* of both block *b1* and *b2*. In consequence, all the activity steps (3) to (6) are executed and the ESM in Figure B1 (b) is violated as all *localWins1*, *remoteWins1* and *remoteWins2* are fired. This will be detected by the Arctis analyzer using model checking. Figure B3 (a) depicts the state space generated by the Arctis analyzer. To facilitate the understanding of the state graph, we added the identifiers of the edges crossing the partition borders. One can see that the traces towards the states 8 and 9 contain a sequence of pins violating the ESM.

3 Mixed Initiative Detection

To identify that a mixed initiative is the actual cause for property violations detected by the Arctis model checker, we investigate the state space further using graph rewriting. According to Floch [Flo03], a mixed initiative is indicated by so-called *mixed initiative states* in which a component may both send and consume signals. The danger of a mixed initiative exists if two interacting components are both in a mixed initiative state at the same time.

In a first step, we label the edges with send and receive labels using the technique explained in [Kra09]. In our example that are M_1, M_2 and M_3 referring to the three partition crossing edges in the activity depicted in Figure B1 (a). By the "!" we describe the sending and by "?" the reception of a communication between the two components. Since M_1 and M_3 show communication from *component 1* to *component 2* and M_2 the other way around, one can see that the states 3 and 11 of the state graph refer to mixed initiative states according to the definition of Floch. In state 3, *component 2* may both send M_2 and receive M_3 while in state 11 *component 1* may send M_3 and receive M_2. By executing the corresponding send actions, both mixed initiative states lead to state 6 which expresses that the two signals forwarding the conflicting initiatives just pass each other. We call it a *conflict state*.

Thereafter, we can check if the traces from the initial node towards the states violating a property always lead via a conflict state. To avoid false positives, we only assume a mixed initiative as the source of errors if all traces to all error states pass at least one conflict state. In our example, the violation of the ESM is detected when reaching the states 8 or 9 and it is easy to see that all traces from the initial state 0 to them pass the conflict state 6.

Technically, we export the state graph created by the Arctis model checker and label the states using the graph transformation tool AGG [Tae04] according to the following rules:

1 Initially, the subgraph contains the error states of the state graph.

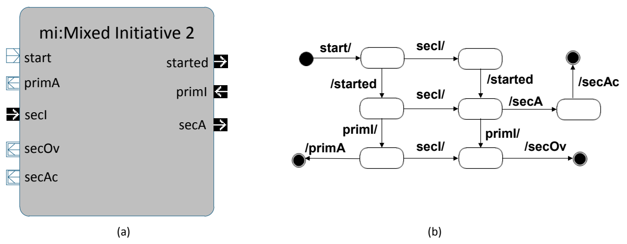


Figure B4. The Arctis block *Mixed Initiative 2*

2 For any vertex in the subgraph that is not a conflict state, we add all its incoming edges as well as their source states to the subgraph.

3 We terminate if we either added the initial state to the subgraph or if all states not yet treated in step 2 are conflict states.

Thus, if the initial node is not in the resulting subgraph, we know that all traces from it to the error states pass a conflict state which gives us advice that an improperly handled mixed initiative might be the source of the errors. For example, in Figure B3 (a) the subgraph consists of the states 6 to 9 and the edges linking them but not the initial state 0 showing that the mixed initiative between the crossing edges M_2 and M_3 are the likely reason for the problem.

The AGG rules use the state space generated by the Arctis model checker as input and are executed automatically without further human intervention. Thus, it is also possible to integrate the algorithm into the Arctis model checker which would enable a seamless detection of mixed initiatives already during the analysis. The integration is planned for one of the next revisions of the model checker.

4 Mixed Initiative Remedy

An established way to deal with errors caused by mixed initiatives is to use prioritization [GY84]. Here, the two conflicting initiatives are marked as *primary* respective *secondary* and, in the case of a conflict, only the primary initiative will take place while the secondary is stopped during communication. In our example, we decided that the initiative M_2 leaving *component 2* shall be the primary one, such that it will always be forwarded to *component 1* while M_3 will be stopped in the case of a conflict. Of course, this prioritization scheme demands a somehow complex logic which, however, can be hidden in a reusable Arctis block as we point out in the following.

4.1 Arctis Blocks handling Mixed Initiatives

Since mixed initiatives are a recurrent phenomenon in reactive distributed software, we created two building blocks providing remedy by prioritization (see [9]) which are available in one of the Arctis libraries. Figure B4 (a) shows one of them. It supports two participants *primary* and *secondary* and arranges that an initiative from the primary one is prioritized against the one of the secondary.

The five pins on the left side of the block are allocated to the secondary and the three on the right to the primary participant. The ESM in Figure B4 (b) describes the behavior realized by the block. It is started from the secondary component by a flow through pin *start* which enables this participant to send its secondary initiative via pin *secI*. Eventually, the start is notified via pin *started* to the primary party who is afterwards enabled to start an own initiative via pin *primI* as long as no secondary initiative passes pin *secA*. If the secondary initiative arrives without being interfered by the primary one, the secondary participant is notified about that via a flow leaving the block through pin *secAc*. If only a primary initiative takes place, the secondary receives it via pin *primA*. If both participants send parallel initiatives via *primI* and *secI*, the secondary is never delivered while the primary one is handed over to the secondary participant via pin *secOv* notifying it that the own one was overridden. The other block in the Arctis library is similar but started from the side of the primary component.

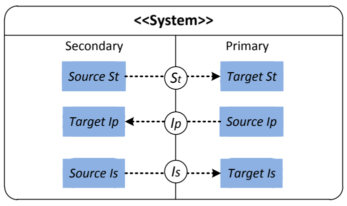


Figure B5. Pattern of a system to be adapted with building block *Mixed Initiative 2*

4.2 Adding the Arctis Blocks

It is demanded that the two Arctis blocks introduced above, have to be started using the pins *start* and *started* before they may handle a mixed initiative. Thus, the system needs a crossing edge between the two partitions that may be redirected in order to act as a starter. To find such an edge, we first analyze the subgraph derived by the algorithm in Sect. 3 and check if there is a crossing edge on all traces towards the mixed initiative that can take that role. Thus, in integrating a mixed initiative block, we have to consider three crossing edges as depicted in the pattern description in Fig. B5. This pattern for the Arctis block *Mixed Initiative 2* (and except of the partition designators also of the other available block) consists of a crossing edge called St describing the starter while I_s and I_p refer to the crossing edges which may cause the mixed initiative error.

In order to allow an adaptation according to the needs of the system engineer, the transformation process is started by asking the engineer for two decisions depending on which a certain group of AGG graph rewrite rules is selected:

1 The engineer has to determine which component should take the role of the primary party in order to identify which of the two mixed initiative blocks needs to be built in.

2 A principle decision about the strategy to handle detected mixed initiatives has to be taken. This reflects, that the two blocks make the occurrence of a mixed initiative visible to the secondary participant and additional functionality handling this case has to be added. Here, we see two different strategies:

   (a) Delete both signals passing each other after transmission. This strategy is sensible when the behavior to be performed by a component is the same irrespective of which party triggered the initiative. In this case, we only have to prevent that this behavior is carried out twice.

   (b) Let the primary initiative prevail and neglect the impact of the secondary one. This solution is useful if both initiatives lead to a different behavior of the involved components.

According to the two decisions, a particular set of AGG graph rewrite rules is selected which execute the integration of a mixed initiative block automatically. If we decide for the strategy to let the primary initiative prevail, however, we face the problem that we need additional functionality to neglect the secondary initiative in case of a conflict. This, however, depends on the particular model, and it is beyond the capabilities of a graph transformation system to decide if any operation executed before passing I_s should also be executed in case of a conflict. To elude this problem, we selected a graph transformation mechanism that renders a correct solution for most functionalities. Nevertheless, it demands that the system engineer looks on the system model resulting from the graph transformation since it is possible that some of the operations have to be rearranged on the local side of the secondary participant. Thus, the graph transformation does not create the correct solution automatically in all cases but we think that it is nevertheless helpful since it reduces a possible manual post-processing to the purely local reordering of single operations which is much easier than the integration of a complex distributed solution from scratch.

At first, the crossing edges corresponding S_t, I_s and I_p in the pattern model are removed and the Arctis mixed initiative block *mi* is added to the model. The remainder of the graph transformation is the connection of the sources respective targets of the removed edges with the pins of *mi* by new edges. For brevity, we describe this process only for the block *Mixed Initiative 2* listed in Fig. B4 as this procedure is similar for the other block. At first *Source St*, i.e., the source node of edge St, will be linked with the pin *start* of *mi* while pin *started* is connected to *Target St*.

On the primary partition, the new wiring of the two conflicting edges I_p and I_s is straightforward since the mixed initiative block disburdens the primary component from any error correction handling. *Source* I_p is coupled with the pin *primI* and pin *secA* with *Target* I_s.

The wiring of the secondary component, however, differs depending on the treatment strategy selected. If both conflicting signals shall be deleted, *Source* I_s is connected with pin *secI* and pin *PrimA* with *Target* I_p. The pins *SecOv* and *SecAc* are not linked at all which according to the robust Arctis semantics means that tokens passing them are deleted.

For the strategy to let the primary initiative prevail, the particular wiring affords to link the pin *primA* with *Target* I_p since this pin reflects that there was no conflict at all. A token passing pin *secOv* contains the data of the primary initiative in the case of a mixed initiative conflict. Since this initiative should prevail, there has also to be made a connection from this pin to *Target* I_p. Actually, the graph transformation rules link both pins *primA* and *secOv* with a newly created merge, i.e., an activity node with at least two ingoing edges but only one outgoing edge to which all incoming tokens are routed. The merge is further connected downstream with *Target* I_p.

The wiring of the vertices and edges specifying the secondary initiative I_s has to consider that its token flow may contain operations necessary for a functionally correct behavior. Operations are another type of UML activities. In Arctis, they are carriers of Java methods which are executed when a token passes. For instance, an operation towards a crossing edge may prepare the transfer format readable by the primary component. After the model modification, such operations shall still be on the path leading to the pin *secI*. Other actions, however, shall only take effect if a conflict does not occur[3] such that they should be linked to the pin *secAc*.

While, as already stated, a general solution to decide about where to place the operations resting on the secondary component before the crossing edge I_s is not possible, we can automate the case that *Source* I_s is a fork node. Here, it is evident that all downstream edges of the fork except for the crossing edge are not relevant for a correct transmission of the secondary initiative. Thus, we can propose a wiring as follows:

1 If *Source* I_s is not a fork, it will be connected with pin *secI* and pin *secAc* will not be linked at all.

2 If *Source* I_s is a fork with two outgoing edges in total, it will be deleted. The source node of its incoming edge will be linked with pin *secI* and pin *secAc* will be connected with the target node of the outgoing edge that is not the crossing edge.

3 If the source node is a fork with three or more outgoing edges, the source node of its incoming edge will also be linked with the pin *secI*. Moreover, we connect the pin *secAc* to the fork such that it is only passed in the case of a successful secondary initiative.

For our button game example, we selected *component 1* as the secondary and *component 2* as the primary partition. Further, we decided to let the primary initiative prevail since, otherwise the pin *remoteWins1* would not be executed in the case of conflict which would violate the ESM of block *Button Game* (see Fig. B1). The result of the graph transformation is depicted in Fig. B6. The modified model produces the state space shown in Fig. B3 (b) such that the ESM of the surrounding block *Button Game* is obeyed. One should mention that the analyzer issues no error but a warning since in the case of a mixed initiative a flow leads to the pin *stop* of block *b1* which is already terminated. This flaw is of no practical relevance as Arctis simply removes tokens in this case what is exactly what we want. So, we do not need any manual re-orderings.

[3]In the button game example, that holds for activity step (5) in Fig. B2 leaving the overall block *Button Game* via the parameter node *localWins1* that should only be triggered if there is no primary initiative at all.

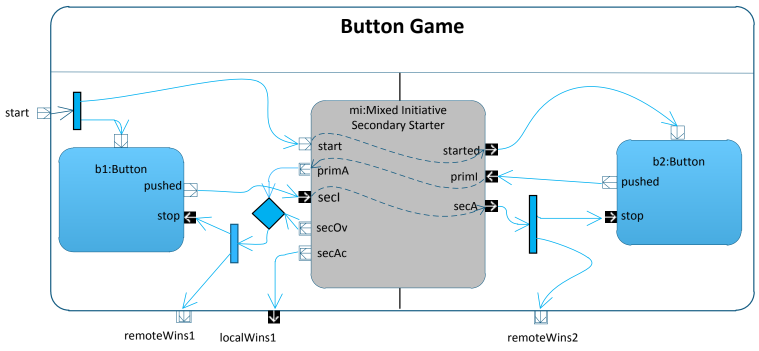


Figure B6. The building block *Button Game* after the transformation

5 Graph Transformation Rules

In this section, we briefly introduce the concept of graph rewriting rules used for the various mixed initiative detection and remedy steps discussed above. As tool-set for the graph transformation we use the Attributed Graph Grammar System[4] (AGG) [Tae04] that, in a flexible way, allows the visually supported creation of rules. Since AGG offers Java APIs, it could be easily integrated into Arctis that is also Java-based.

The transformation rules mainly consist of two parts. A pre-pattern describes a graph pattern that has to be replaced while the corresponding post-pattern models the result of the replacement. Moreover, a rule may contain additional conditions to constrain when it may be applied. The input to a rule is a so-called host-graph which, by replacing the part matching the pre-pattern by the post-pattern, will be transformed to a post-graph.

Altogether 22 rules are used to detect mixed initiatives and to add one of the two Arctis mixed initiative blocks to a UML activity-based system model. For the sake of brevity, we list only the different categories of rules and provide a closer description of only one rule while the others can be looked at on the WWW.[5] The rules can be structured in three groups:

1 Label the states of an Arctis state graph to detect a mixed initiative as discussed in Sect. 3.

2 Search for the pattern described in Fig. B5 of the state graph to create a subgraph with labels. During this stage the crossing edges I_p, I_s and S_t are marked.

3 Insert the selected Arctis mixed initiative block and wire it with its environment as described in Sect. 4.2.

Figure B7 shows a rule of group 3 that is used to add the block *Mixed Initiative 2* to an activity. On the left side, the pre-pattern is depicted. It contains the classes 3 and 4 of type *role* which refer to the two partitions of the activity. These two constructs are applied to enhance the collaboration description used in Arctis to model the relation between blocks and particular components. The other constructs refer to the activities to be amended. In particular, the three labeled cycles refer to the crossing edges involved while label 1:f describes the one to be used to start the block. Label 7:f refers to the primary and 8:f to the secondary initiative. The post-pattern is shown on the right side. The collaboration is extended by a new collaboration use that corresponds to the added mixed initiative block as well as to the links to the two components. The activity is supplemented by a call behavior action, i.e., an Arctis block, as well as references to its pins. The various edges of the pre-pattern are now replaced by others linking the

[4]We currently replace AGG by Henshin [ABJ+10] which is more flexible and supports the Eclipse Modeling Framework (EMF) also used by Arctis.
[5]http://www.item.ntnu.no/people/personalpages/phd/simon/start

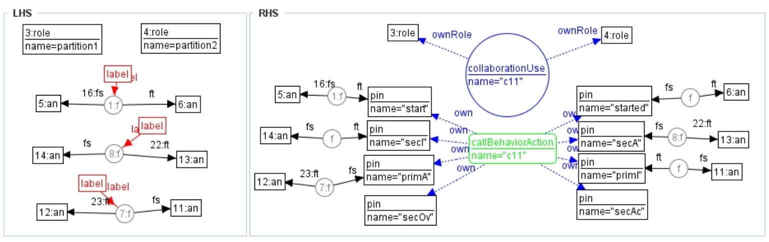


Figure B7. Rule inserting the block *Mixed Initiative 2*

original source respective target nodes with the appropriate pins of the mixed initiative blocks. This rule is used for both strategies mentioned in Sect. 4.2. It renders the final result if we want to delete both mixed initiatives. If we like to let the primary initiative prevail and neglect the secondary one, it creates an intermediate system model which will be further amended by other rules.

6 Related work

In visual language-based specification techniques like the UML, graph grammar techniques are more and more utilized. For instance, in [WTEK08] Winkelmann et al. translate restricted OCL constraints into equivalent graph constraints which enables an automatic generation of instance models from the OCL meta-models. Gronmo and Møller-Pedersen propose so-called aspect activity diagrams that extend activity models by aspect-oriented weaving [GM08]. Likewise, Mussbacher et al. use an extension of the User Requirement Notation (URN) to weave in aspects [MWA10]. A difference to our approach is that both techniques demand for explicit syntax extensions to define aspect orientation concepts like point cuts, which makes the understanding of the models more complicated. In [HHR+11], Hegedüs et al. use graph grammars as the fundamental technique of the framework to generate quick fixes of business flows specified in the Business Process Model and Notation (BPMN). Like our work, this approach uses graph transformation for the remedy of errors, albeit on a more abstract modeling level. Our work is also similar with [LK10] who use graph transformation rules to slice UML models using transformation rules. The difference is that Lano and Kolahdouz-Rahimi concentrate on the slicing of state machines.

Graph grammar systems are further used to support system development in specific domains. Mens et al. [MVDJ05] use graph transformation to formalize refactorings of software. Bucchiarone et al. [BPVR09] give a formal definition of self-adaptiveness and self repair systems based on the T-typed hyper-graph grammar system. They also use AGG to model and verify the hyper-graphs. In [JWEG07], critical pair analysis is used to detect the dependencies and conflicts between features of a Software Product Line (SPL). Domain specific concerns are also addressed by particular patterns. For instance, in [JPW02] security patterns are abstracted from model-based system development and specially treated in security-critical systems.

7 Concluding Remarks

We introduced the use of graph transformation for the detection and remedy of mixed initiative conflicts, a particular kind of development errors in distributed systems. The approach is highly automatic and promises to support the engineer significantly in error recovery and remedy. The set of rules was also applied to a wake-up call scenario [9] as it is used in hotels. Again we received a correct result that did not need further manual corrections.

To generalize this experience, we have also to guarantee that the model transformation does not introduce new errors. In particular, we must assure that the new wiring complies with the properties of the integrated mixed initiative block. Further, we have to prove that the transformed model is consistent with the original one. Of course, as desired, the model transformation changes the system behavior to solve the conflict but it should not be changed in the ordinary case that only one initiative takes place at a time.

These two questions can be verified in temporal logic which, however, cannot be shown here due to the space limit.

There are other system layouts which might lead to mixed initiatives, e.g., three or more components may use ring-shaped communication such that there are no conflicting crossing edges between any two of them. We want to find out more system patterns that indicate mixed initiatives and create corresponding graph transformation rules to alleviate them. Further, we intend to use the approach for the detection and remedy of other kinds of errors.

Graph transformation seems also promising to add security protection against malicious attacks. While the integration of counter-measures, in general, is complex and tedious, in some cases it can be done with limited human guidance such that graph transformation is the appropriate means. An example is [GKH11] introducing the structured integration of security mechanisms to protect sensible communications against wiretapping.

The work introduced above was the second approach utilizing graph transformation in SPACE. Before, we used this technique to transform flow-global choreographies, a more abstract modeling technique, to Arctis models [HKL+11]. Both approaches profit from the fact that it is much easier to create and change a set of graph rewrite rules than to develop a model transformation tool doing the same model changes manually. This allows for an easy adaptation of engineering tools for visual languages to new challenges arising during deployment.

A strength of Arctis is that it supports reuse of sub-models in certain application domains. In average, 70% of a system model consist of building blocks reused from previous projects (see [18]). Utilizing the flexibility of graph rewriting, one can complement the domain-specific libraries of Arctis blocks with sets of graph transformation rules such that an engineer is not only provided with suitable sub-models but also with a convenient functionality to deal with them.

Acknowledgments The research was carried out under the research and development project "Infrastructure for Integrated Service" (ISIS) funded by the Research Council of Norway.

Bibliography

[ABJ+10] T. Arendt, E. Biermann, S. Jurack, C. Krause, G. Taentzer. Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In Petriu et al. (eds.), *Proceedings of the 13th Int. Conference on Model Driven Engineering, Languages and Systems (Models)*. LNCS 6394, pp. 121–135. Oslo, 2010.

[BH93] R. Bræk, Ø. Haugen. *Engineering Real Time Systems — An Object Oriented Methodology using SDL*. Prentice Hall, 1993.

[BPVR09] A. Bucchiarone, P. Pelliccione, C. Vattani, O. Runge. Self-Repairing systems modeling and verification using. In *Software Architecture, 2009 European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*. Pp. 181–190. 2009.

[Flo03] J. Floch. *Towards Plug-and-Play Services: Design and Validation using Roles*. PhD thesis, Department of Telematics, Norwegian University of Science and Technology (NTNU), 2003.

[GKH11] L. Gunawan, F. A. Kraemer, P. Herrmann. A Tool-Supported Method for the Design and Implementation of Secure Distributed Applications. In *Engineering Secure Software and Systems*. LNCS 6542, pp. 142–155. Springer, 2011.

[GM08] R. Grønmo, B. Møller-Pedersen. Aspect Diagrams for UML Activity Models. In Schürr et al. (eds.), *Applications of Graph Transformations with Industrial Relevance*. Pp. 329–344. Springer-Verlag, Berlin, Heidelberg, 2008.

[GY84] M. G. Gouda, Y.-T. Yu. Synthesis of Communicating Finite State Machines with Guaranteed Progress. *IEEE Transactions on Communications* 32(7), July 1984.

[HHR+11] Á. Hegedüs, Á. Horváth, I. Ráth, M. Branco, D. Varró. Quick fix generation for DSMLs. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE Computer, Pittsburgh, PA, USA, 2011.

[HKL+11] F. Han, S. B. Kathayat, H. N. Le, R. Bræk, P. Herrmann. Towards Choreography Model Transformation via Graph Transformation. In *IEEE Conference on Software Engineering and Service Science, ICSESS 2011*. IEEE Computer, Beijing, 2011.

[JPW02] J. Jürjens, G. Popp, G. Wimmel. Towards Using Security Patterns in Model-based System Development. 2002.

[JWEG07] P. K. Jayaraman, J. Whittle, A. M. Elkhodary, H. Gomaa. Model Composition in Product Lines and Feature Interaction Detection Using Critical Pair Analysis. In Engels et al. (eds.), *MoDELS*. LNCS 4735, pp. 151–165. Springer, 2007.

[KH07] F. A. Kraemer, P. Herrmann. Transforming Collaborative Service Specifications into Efficiently Executable State Machines. *ECEASST* 6, 2007.

[KH09] F. A. Kraemer, P. Herrmann. Automated Encapsulation of UML Activities for Incremental Development and Verification. In Schürr and Selic (eds.), *Proceedings of the 12th Int. Conference on Model Driven Engineering, Languages and Systems (Models), Denver, Colorado, USA, October 4-9, 2009*. LNCS 5795, pp. 571–585. Springer-Verlag Berlin Heidelberg, 2009.

[KH10] F. A. Kraemer, P. Herrmann. Formal Techniques for Distributed Systems, Joint 12th IFIP WG 6.1 International Conference, FMOODS 2010 and 30th IFIP WG 6.1 International Conference, FORTE 2010, Amsterdam, The Netherlands, June 7-9, 2010. Proceedings. In Hatcliff and Zucca (eds.), *Formal Techniques for Distributed Systems*. LNCS 6117. Springer, 2010.

[Kra09] F. A. Kraemer. Automatic Generation of Compatible Interfaces from Partitioned UML Activities. In Reed et al. (eds.), *SDL 2009: Design for Motes and Mobiles*. LNCS 5719, pp. 182–199. Springer Berlin / Heidelberg, 2009.

[Kra11] F. A. Kraemer. Engineering Android Applications Based on UML Activities. In Whittle et al. (eds.), *Model Driven Engineering Languages and Systems*. LNCS 6981, pp. 183–197. Springer Berlin / Heidelberg, 2011.

[KSH07] F. A. Kraemer, V. Slåtten, P. Herrmann. Engineering Support for UML Activities by Automated Model-Checking — An Example. In *4th International Workshop on Rapid Integration of Software Engineering Techniques (RISE)*. 2007.

[KSH09] F. A. Kraemer, V. Slåtten, P. Herrmann. Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services. *Journal of Systems and Software* 82(12):2068–2080, 2009.

[LK10] K. Lano, S. Kolahdouz-Rahimi. Slicing of UML Models Using Model Transformations. In Petriu et al. (eds.), *Model Driven Engineering Languages and Systems*. LNCS 6395, pp. 228–242. Springer Berlin / Heidelberg, 2010.

[MVDJ05] T. Mens, N. Van Eetvelde, S. Demeyer, D. Janssens. Formalizing refactorings with graph transformations: Research Articles. *J. Softw. Maint. Evol.* 17:247–276, 2005.

[MWA10] G. Mussbacher, J. Whittle, D. Amyot. Modeling and detecting semantic-based interactions in aspect-oriented scenarios. *Requirements Engineering* 15:197–214, 2010.

[Obj10] Object Management Group. Unified Modeling Language: Superstructure, Version 2.3. 2010.

[SDW08] C. Shin, A. K. Dey, W. Woo. Mixed-initiative conflict resolution for context-aware applications. In *UbiComp'08*. Pp. 262–271. 2008.

[Tae04] G. Taentzer. AGG: A Graph Transformation Environment for Modeling and Validation of Software. In Pfaltz et al. (eds.), *Applications of Graph Transformations with Industrial Relevance*. LNCS 3062, pp. 446–453. Springer, 2004.

[WTEK08] J. Winkelmann, G. Taentzer, K. Ehrig, J. M. Küster. Translation of Restricted OCL Constraints into Graph Constraints for Generating Meta Model Instances by Graph Grammars. *Electronic Notes on Theoretic Computer Science* 211:159–170, 2008.

PAPER C

Modeling and Verifying Real-time Properties of Reactive Systems

Fenglin Han, Peter Herrmann and Hien Le

Proceedings of the 18th International Conference on Engineering of Complex Computer Systems (ICECCS 2013). Singapore, 2013.

MODELING AND VERIFYING REAL-TIME PROPERTIES OF REACTIVE SYSTEMS

Fenglin Han
Norwegian University of Science and Technology,
Trondheim, Norway
sih@item.ntnu.no

Peter Herrmann
Norwegian University of Science and Technology,
Trondheim, Norway
herrmann@item.ntnu.no

Hien Le
Norwegian University of Science and Technology,
Trondheim, Norway
hiennam@item.ntnu.no

Abstract SPACE is a model-driven engineering technique for reactive distributed systems. It enables to develop system models from reusable building blocks, formal analysis by model checking as well as automated transformation to executable code. In this paper, we describe an extension of the SPACE formalism which allows to model and verify also real-time behavior. In particular, one specifies real-time constraints in the interface descriptions of the building blocks, so-called Real-Time External State-Machines (*RTESMs*). The RTESMs are translated to guards, clocks and invariants of Timed Automata which can be analyzed by means of the model checker UPPAAL. The approach is explained by a component protecting an electrical motor controller system against overspeed. In particular, we prove that by keeping certain maximum response times, this system guarantees that the speed of the motor stays within certain limits.

1 Introduction

Model-based engineering is considered as practical to create high-quality distributed software since it enables stepwise development with varying degrees of abstraction as well as simulation, verification and evaluation. This is of particular importance for the design and verification of real-time embedded software used in safety critical systems. For instance, Buttazzo [1] claims that real-time systems are more vulnerable than other kinds of systems and names three aspects facilitating the design of high quality software: The design of software before building it, complexity reduction, and the enforcement of system compatibility within the design.

The engineering technique SPACE [2] and its tool-set Arctis [22] make a model-driven development process supporting these three issues possible. System behavior is modeled by UML 2 activities (see [4]) that, in a Petri net-like fashion, express behavior by tokens flowing via the edges of a graph [5]. The approach is scalable since an activity may contain *building blocks* that each represents an own activity. The tokens may jump between the two activities, to which a building block refers. The building blocks support also the reuse of sub-models since they allow to model recurrent behavior once and to use the resulting building blocks later in various application models. As discussed in [16], in average 70% of a system model can be developed by reusing blocks from the Arctis libraries. The easy reuse of building blocks is supported by External State Machines (ESMs) [16] which describe the interface behavior of the building blocks. Thus, a user who wants to integrate a building block into a system model, does not need to understand its internal behavior, i.e., the activity it refers to, but only its ESM. Further, the ESMs are an effective means to mitigate the state space explosion problem which may occur when model checking that a system model fulfills certain properties [22]. They make it possible to replace most of the activities by the more abstract ESMs in the model checker runs which reduces the state space to be checked effectively. Besides analysis by model checking, Arctis allows to automatically translate a system model into executable code. Currently, it supports the generation of Java code but extended versions for C resp. C++ are under development.

Our previous work has been centered on modeling functionally correct applications without considering real-time properties. In particular, we have used a self-developed model checker to verify basic functional properties (e.g., whether a building block complies with its ESM) while we apply the model checker TLC [7], which is based on Lamport's Temporal Logic of Actions [5], for more complex proofs (see [27]).

In this paper, we discuss how the advantages of a modular specification and verification in SPACE can be exploited for the development of real-time systems. In particular, we introduce an extension of the ESMs to so-called Real-Time ESMs (RTESMs) that make the specification of certain real-time constraints possible. This allows to specify the real-time constraints, a building block requires from its environment as well as the real-time properties guaranteed by itself. Moreover, we added UPPAAL [11] to the set of model checkers supported by Arctis and realized an automatic transformation of each building block and RTESM into timed automata [11] that models relevant time constraints like state invariants, transition guards and clock updates. The real-time properties stated in the RTESMs are translated into formulas of the temporal logic *Timed Computation Tree Logic* (TCTL) [8]. Together the building blocks of a system model form a network of timed automata that can be analyzed by UPPAAL for meeting the TCTL formulas.

The article is arranged as follows: Section 2 presents the SPACE method. For that, we introduce a building block of a real-life embedded system. In Sect. 3, we sketch the Timed Automata and their verification with UPPAAL. Further, the RTESMs are presented. Section 4 discusses the compositional character of the real-time behavior verification and presents the results of proving the real-time aspects in our example. We verify in Sect. 5 that the compositional model checks are sufficient to guarantee that the real-time properties are also met by the overall system. The paper is completed by references to related work in Sect. 6 followed by some concluding remarks.

2 Arctis Building Block Model

In this section, we introduce the model-based development approach using SPACE and Arctis by showing a component of a control system for electrical motors which has been developed by Asea Brown Boveri, Ltd. (ABB).

The modeling and verification includes a list of functional units and test cases of a motor controller system, which cooperate with each other realizing the start of the control software. We ignore the big view of system collaboration and concentrate on the Safety Limited Speed System (SLS) component. The SLS complies with the safety standard IEC 61800-5-2 [15] in order to guarantee that the speed of a motor stays below a configurable maximum limit. To achieve that, this component is able to reduce the motor speed or even initiate the stopping of the motor if necessary.

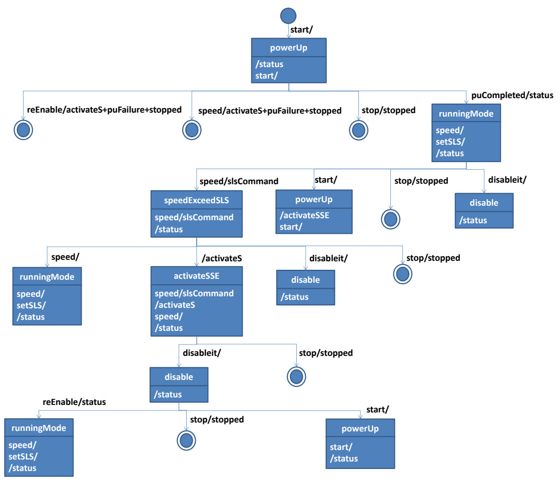


Figure C1. The External State Machine (ESM) of the SLS block

2.1 Interface of the SLS Building Block

To understand the functionality of the SLS block, we look first on its External State Machine (ESM) which is depicted in Fig. C1.[1] The ESM reflects nicely that, according to standard IEC 61800-5-2, the SLS has six different control states. Thereby, the starting (•) and termination (◉) nodes of the ESM refer to the state *idle* while the other states are represented by vertices containing the corresponding state names:

- *idle:* The motor control system is switched off.
- *powerUp:* The motor control system is starting.
- *runningMode:* The motor is running normally, i.e., it is below its maximum speed limit.
- *speedExceedSLS:* The motor runs above its permitted speed limit but did not yet exceed the maximum time period after which it has to be shut down by executing the Safe Stop Emergency (SSE) handler, another system component, that manages the execution of emergency stops of the motor.
- *activateSSE:* The SSE handler was triggered and the motor was shut down.
- *disable:* The SLS block is disabled after the power for the motor was removed and it cannot produce any torque again.

The transfer of tokens between the activity of a building block and its environment is modeled by so-called pins. That are syntactical constructs used in both the activity modeling the behavior of the block and the one using it. The ESM of a building block models the interface behavior of a block by linking each ESM transition to a number the block's pins. The activity to which the block refers, may only carry out a sequence of token flows if the pins passed in this sequence are exactly those linked to an ESM transition that is executable in the current ESM state. For example, the initial transition from state *idle* to state *powerUp* must only be carried out if a token passes the pin *start*. Transitions not changing the state of the ESM are listed in the state identifiers, e.g., in state *powerUp* two transitions caused by tokens passing one of the pins *status* or *start* may be executed without changing the ESM state.

In the transition markings, a pin identifier before the slash symbol (e.g., *start* /) marks that the transition is triggered by a token flow originating from the environment of the building block. In contrast, if

[1]To avoid the problem of a proper vertex placement when displaying an ESM, Arctis uses the hierarchical style listed in Fig. C1.

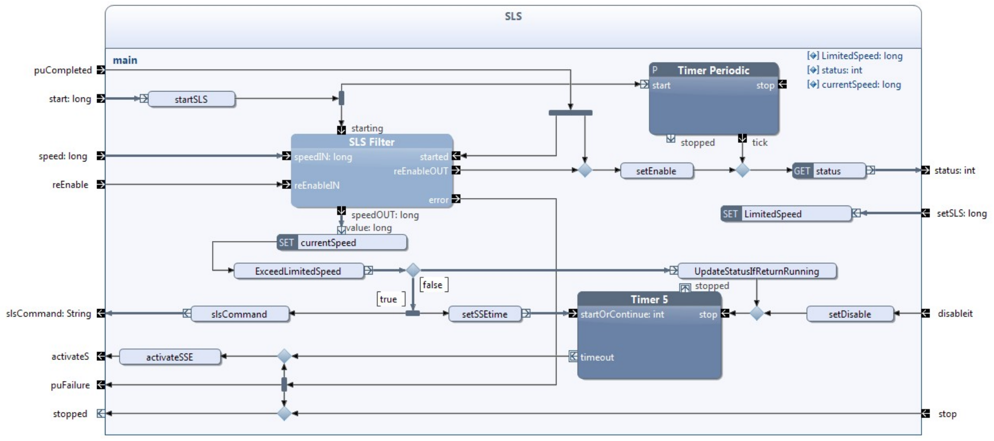


Figure C2. Activity of Secure Limited Speed (SLS) Building Block

there is no pin ahead of the slash (e.g., */ status*), the transition is initiated by the block itself which starts a flow towards its environment.

A combination of different pins on a single transition specifies that several pins are passed by token flows in the same transition. For instance, *speed / activateS + puFailure + stopped* expresses that the transition is triggered by an incoming token passing the pin *speed* which in the same step leads to tokens leaving the block via the pins *activateS*, *puFailure* and *stopped*.

Altogether, our block uses 12 different pins to interact with its environment. They are sketched in the following:

- *start:* The start sequence of this block is initiated. The token uses a long integer as parameter which describes the maximum speed limit of the motor.
- *puCompleted:* A token passing this pin confirms that the powering up phase of the motor is finished.
- *speed:* The current speed of the motor is sent as a long integer value in the tokens passing this pin.
- *slsCommand:* A token leaving the SLS block through this pin contains a string which is a command to the motor to reduce its speed.
- *activateS:* A token passing this pin activates the Safe Stop Emergency (SSE) function stopping the motor.
- *puFailure:* By this pin, the environment of the SLS is notified of a failure in the powering up phase due to an incoming speed or re-enabling signal.
- *status:* Tokens passing this pin send status information about the SLS block in form of integer values.
- *setSLS:* By tokens passing this pin, the maximum speed limit may be altered. The tokens contain a long integer carrying the new maximum speed.
- *disableit:* This pin is used to disable the SLS block.
- *reEnable:* A token through this pin re-enables the SLS block.
- *stop:* Flows passing this pin switch off the SLS block.
- *stopped:* This pin is a notification that the SLS block is terminated.

The block is started by a token passing pin *start* and remains in state *powerUp* until a notification is received via *puCompleted*. Thereafter, the system is in state *runningMode* in which the current speed is received periodically. If the speed exceeds the given speed limit, the SLS block switches to state *speedExceedSLS* which leads to control commands via pin *slsCommand*. The block stays in the state *speedExceedSLS* until either the speed falls below the limit again after which it is set to state *running-Mode* or a timeout happens. In the latter case, a token is sent via *activateS* to initiate an emergency stop and the SLS block is set to the state *activateSSE*. Further, the system may be disabled as indicated by tokens via pin *disableit*. Thereafter, the block will be in state *disable* until it is re-enabled via pin *reEnable* and reaches state *runningMode*. In addition, the SLS notifies its environment about its status by flows through pin *status*. It is terminated either by an initiative from the environment which may send tokens via pin *stop* or by erroneous speed or re-enabling signals during the power up phase. In the latter case, the environment is also notified by a failure message via pin *puFailure* and an emergency stop command via *activateS*.

2.2 Behavior of the SLS Component

As sketched in the introduction, system and building block behavior is modeled in SPACE and Arctis by UML 2 activities [4]. The activity of the SLS block is depicted in Fig. C2. It contains the blocks *Timer Periodic* and *Timer 5* which were taken from an Arctis standard library and describe two different kinds of timers. The block *SLS Filter* was created by the developer of the SLS block to handle the special treatment of speed and re-enabling messages occurring in the power up phase. *SLS Filter* is a so-called *shallow block* for which only the pins and the ESM have to be specified while Arctis generates the block behavior automatically.

The 12 pins introduced above are specified as *parameter nodes* on the edge of the activity in Fig. C2 that we will also call "pins" for simplicity. A flow passing pin *start* is forwarded to the operation action *startSLS* that contains a Java method of the same name. In this method, the limited speed value is stored in the long integer variable *LimitedSpeed* while the block status, represented by the correspondent variable, is set to the value *active*. Thereafter, the token proceeds to a fork node in which it is duplicated. One token copy reaches the block *SLS Filter* to enable error handling of speed and re-enabling messages during power up. The other copy starts the block *Timer Periodic*. From now on, this block will, in intervals, create tokens leaving its pin *tick* and being forwarded via a merge node to the get variable action *GET status*. Here, the current value of variable *status* is stored as a token parameter. Afterwards, the tokens are sent to the environment via pin *status*. The completion of the powering up phase is notified by a token entering the block via pin *puCompleted*. This token is duplicated in a fork. One copy switches the block *SLS Filter* into normal mode such that speed and re-enabling messages are normally treated while the other copy adjusts the status variable and issues a status output to the environment.

Speed messages reaching pin *speed* are forwarded to the block *SLS Filter*. If the block is in the power up phase, the token is forwarded via pin error to a fork which generates three copies forwarded to the pins *activateS* via an operation adjusting the status, *puFailure* and *stopped*. The pin *stopped* is a terminating pin. When it is passed by a token, all remaining tokens in the system are removed and the three inner blocks are set to their respective idle states. Thus, the SLS building block is re-initialized. If the power up phase is already completed, the speed message is forwarded from block *SLS Filter* to the set variable action *SET currentSpeed* storing the parameter of the token. Thereafter the flow reaches the operation *ExceedLimitedSpeed* in which the current speed is compared with the speed limit. The token leaving this block carries a boolean value. If the motor is on overspeed, it is forwarded from the decision node via the edge *true* to a fork. One of the copies is forwarded to the operation *slsCommand* which generates the speed reduction command leaving the block via pin *slsCommand*. The other token starts block *Timer 5* to enable an activation of the SSE system if the overspeed stays for a certain period of time, i.e., 1000 ms. If the motor runs in normal speed, the token leaves the decision via the edge *false* which updates the status and stops the timer. A timeout of the timer in block *Timer 5* leads to a token passing pin *activateS*.

Disabling the SLS block by sending a token through pin *disableit* switches off *Timer 5* since an activation of the SSE is not allowed in the state *disable*. The block can be enabled again by a flow via pin *reEnable* which during power up leads to an error handling while, otherwise, the block is re-enabled followed by a status message. Further flows enable to change the limited speed via a token passing pin *setSLS* and to stop the SLS building block using the pins *stop* and *stopped*.

Using the built-in Arctis model checker [22], we could easily prove that the activity listed in Fig. C2 fulfills the ESM depicted in Fig. C1 such that the block correctly realizes its interface behavior. This block can now be combined with similar blocks of the motor control unit and, utilizing the Arctis code generation, program code can be created.

3 Real-Time Extension

The two key performance requirements of the Safety Limited Speed (SLS) component can be stated as follows:

1 Except for the idle and power up phases, a reduce speed command (*slsCommand*) should always be executed when a speed notification showing overspeed of the motor is detected.

2 The activation of the Safety Stop Emergency (SSE) component shall be executed not later than 1000 ms after overspeed was detected, as long as the speed does not fall back below the limit in this period.

The first requirement holds obviously since one can easily see from the activity in Fig. C2 that every speed input showing overspeed leads to a slsCommand. Formally, that can be verified using the model checker TLC [7].

The second requirement describes a typical hard real-time requirement that, before now, could not be modeled and verified directly by the tool-set Arctis. To prove properties like this one, we had to extend the SPACE syntax and the analysis capabilities of Arctis in a way that real-time properties including bounded response-time guarantees can be modeled and verified as well (see also [16, 15, 16]). While the semantics of SPACE is based on Lamport's Temporal Logic of Actions (TLA) [5], we refrained from using the corresponding way to model real-time [17, 10]. The problem is that in this method, real-time is specified by real numbered values which would lead to systems enclosing huge (or even infinite) numbers of states that exceed the ability of TLC and other model checkers. Thus, we would need to use either manual verification or rely on theorem provers requiring a significant verification guidance such that carrying out the proofs would be long-lasting and tedious. Therefore we decided to base the system extension on Timed Automata [11] and the verification tool UPPAAL [11] which are sketched in Sect. 3.1. We decided to extend the ESMs by annotations that make it possible to model real-time properties which have to be kept by both the building block to which the ESM belongs and its environment. The resulting *Real-Time External State Machines* (RTESMs) are introduced in Sect. 3.2. Together with the activities, they are transformed to Timed Automata which form the input for the UPPAAL-based proofs. The corresponding mapping is described in Sect. 3.3.

3.1 Timed-Automata and UPPAAL

Several approaches to model real-time properties are available, e.g., IO automata [19], hybrid automata [20] and timed statecharts [21]. We decided to apply *Timed Automata* (TA) [11] which were employed by Rajeev Alur and David Dill in 1990 since TAs fit excellently with SPACE and Arctis and provide a powerful model checking environment as we will discuss below. Timed Automata are extended finite state machines which allow to specify real-time values as *environment clock variables*. These variables are synchronized with the clock such that they express natural time elapsing. Each variable may be set or reset when the state machine carries a certain transition. Further, one may define so-called *clock invariants* restricting the time, a state machine may rest in a certain state of a TA.

UPPAAL [11] is a modeling and verification tool for Timed Automata which is suited to verify that a system fulfills certain real-time properties. It enables to express systems consisting of various timed state machines, so-called *templates*, which can run in parallel and interact via synchronization channels. A TA transition can be amended by the following annotations:

- *Guards:* A transition may only be taken if its guard is true.

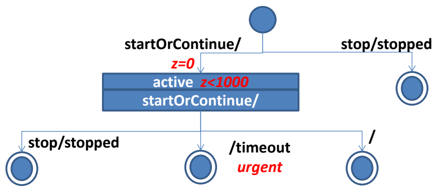


Figure C3. Real-Time External State Machine (RTESM) of block *Timer 5*

- *Synchronization:* Timed automata exchange signals with each other synchronously by *send* (!) and *receive* (?) signal pairs. The synchronization supports binary transmissions from a timed automaton to another one as well as broadcasts. In the latter case, a signal is sent by one timed automaton and received by various others.
- *Updates:* The environment variables are updated when a transition is carried out.

The clocks, invariants, guards, and updates of the timed automata are represented in UPPAAL as annotations of the state machines using a C-like syntax. Further, timed properties to be verified are expressed as formulas in a subset of the branching-time temporal logic *Timed Computation Tree Logic* (TCTL) [8].

3.2 Real Time External State Machines

We extend the ESMs in SPACE to *Real-Time External State Machines* (RTESMs) by introducing also environment clock variables, clock invariants and updates. This well-arranged concept of the Timed Automata allows to model real-time properties to be fulfilled by the interface behavior of building blocks in an easily understandable way. Like in the TAs, the states may contain clock invariants determining how long the RTESM may stay in a state. However, the initial and terminating states must not contain clock invariants. The RTESM transitions may contain updates to manage the environment variables.

As an example, we depict the RTESM of the Arctis building block *Timer 5* in Fig. C3. This block realizes a persistent timer running 1000 ms after being started via a token through pin *startOrContinue* until it issues a token via pin *timeout*. The time is expressed by the clock variable z which is set to its initial value 0 when the transition *startOrContinue* is executed for the first time switching the RTESM from the state *idle* to *active*. When further tokens pass the pin *startOrContinue* in state *active*, z is not set to 0 which models the persistence property of the timer. The state *active* is provided with the clock invariant $z < 1000$ stating that the RTESM may only stay less than 1000 ms in this state before it has to be left, i.e., an *urgent* action (*timeout*) has to be taken when the timer reaches 1000 ms. Transitions representing urgent actions are marked with the label *urgent*. These constructs allow to model bounded liveness properties [16], e.g., guaranteeing that the transition *timeout* must be executed if it, otherwise, will be continuously enabled. In Sect. 4, we show that the urgent transitions are used to find out whether a clock invariant is guaranteed by the activity of the building block to which the RTESM is assigned or by the one carrying the block identifier. To guarantee that this is unambiguously defined, each state containing a clock invariant must have at least one downstream edge marked as urgent and all urgent edges leaving the same source state have to be either triggered by the activity in the block or the one modeling the environment.

The RTESM of the block *Timer 5* states that, as long as the timer is not stopped (transition $stop/stopped$) resp. deleted together with the block including it (transition $/$), a token has to pass pin *timeout* before the 1000 ms limit is reached. That is exactly the real-time property we like to be guaranteed by the building block.

3.3 Mapping from SPACE to UPPAAL

The reactive SPACE semantics [5] defines the UML 2 activities as state transition systems in a run-to-completion fashion. The states are represented by tokens resting on activity vertices and edges. A token may stop only on activity nodes modeling system elements demanding it to wait for a certain period of

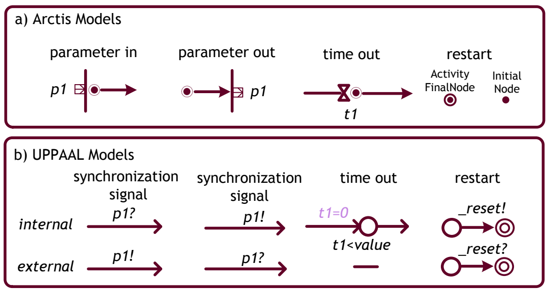


Figure C4. Mapping between the Activity Behavior and TAs

time. Besides initial nodes describing the token setting at system start, there are receiving nodes at which a token has to wait for a signal from an external event source. Likewise, tokens reaching timer nodes have to wait for a fixed perioded of time. Further, a token may wait on the edge leading to a join node which, being the complement of a fork, synchronizes various tokens, i.e., only when tokens are reaching all of its input edges, one of them may pass the join. In analogy to Petri-nets, we call the nodes and edges, on which tokens may rest, *inner places*. At most one token may wait at the same time on an inner place.

A collaborative building block as discussed in [22] refers to various components which are represented in its activity by different partitions. Thus, edges crossing a partition border model the interchange of signals between the components. Since the communication is asynchronous, these edges include *queue places* on which tokens rest during transfer. In contrast to the inner places, queue places may contain various tokens at the same time. They are stored in a FIFO queue with a limited queue size.

In the SPACE semantics, a transition corresponds to a so-called activity step, in which a token starts at an inner or queue place and moves forward on the activity graph in a run-to-completion fashion until it reaches another place respectively a final node on which it is deleted. If the token passes an operation, the corresponding Java method is executed. If the pass of the token goes via a fork node, on which it is duplicated, all copies are handled within the same activity step. Likewise, the jumping of a token between activities via the pins takes place within a single activity step. The reactive nature of the semantics is guaranteed since any transition is triggered by the reception of a signal or a timeout. The semantics facilitates an automatic transformation of the activities to executable UML 2 state machines [17] from which Java code can be generated.

By our Arctis to UPPAAL mapping algorithm introduced below, an Arctis building block is transformed into a network of Timed Automata (TAs). To facilitate the understanding of this transformation, we distinguish so-called *internal TAs*, which are generated from the activities, from *external TAs* transformed from the RTESMs. The states of an internal TA correspond to the different token settings on the queue and inner places of an activity while the activity steps are mapped to the TA transitions. Some aspects of the mapping of Arctis activities to internal TAs are highlighted in Fig. C4:

- A token arriving at an activity via pin *p1* is translated to a binary synchronization channel in which the internal TA carries out a receive signal (*p1?*).
- A token leaving an activity via pin *p1* is mapped to a binary synchronization channel in which the internal TA executes a send signal (*p1!*).
- A timer *t1* in an activity is transferred to a node of the internal TA, which is provided with an environment clock variable that is set to 0 by the upstream edge to this node. Further, a clock invariant states that the TA may only be in the state if the upper bound limit of the clock is not yet reached (expressed as $t1 < duration$).
- A terminated building block (e.g., by a token reaching an Activity Final Node ($\odot$), may be restarted at any time. To model this property also in the TAs, we add special reset transitions from the final node to the initial node using a broadcast channel $\_reset$.

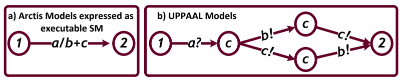


Figure C5. Several pins in an RTESM transition

In practice, a UML activity is first transferred into an executable state machine using the Arctis transformation tool. Thereafter, this state machine is automatically transformed to the internal TA following the mapping sketched above.

Similar to the TAs, the RTESMs are state transition systems which allow for a direct mapping of the states including the clock invariants. The RTESM transitions are transformed to send and receive signals of the synchronization channels representing the pins with which they are annotated. As shown in Fig. C4, all token flows leaving a building block B are specified in the external TAs modeling B's own RTESM and the ones of its inner blocks as receive signals (*p1?*). This is the case since, from B's view, the RTESMs represent the activities of its environment respectively the activities of the inner blocks which all receive the tokens sent by B. Likewise, token flows heading towards B are modeled in the external TAs by send signals (*p1!*).

An issue to be treated when mapping RTESM transitions to transitions of the external TAs, is that the RTESM transitions may refer to various pins while TAs do not allow to combine different synchronization channels in a single transition. We address that by sequences of TA transitions following the order of the pins. This leads to intermediary states that are declared as *committed locations*, i.e., states not modeling any passing of time. For pins which may be executed in parallel, we provide separate paths enabling any communication order. According to our experience, the additional states do not have an appreciable impact on the model checker performance.

An example is shown in Fig. C5. Here, due to the RTESM transition, a building block receives first a token via pin a which, in the same activity step, leads to two tokens leaving the block via b and c in parallel. In the corresponding external environment TA, first a receive signal via channel a is received which leads to an interleaving of sending signals via b respective c. Finally, like in the internal TAs we use special reset transitions from the final to the initial nodes to model that RTESMs can be restarted at any time.

As an example of mapping an RTESM to an external TA, we use the RTESM of block *Timer 5* depicted in Fig. C3 and the resulting external TA shown in Fig. C6. We see that the three states of the RTESM. i.e., *active* as well as the initial and final states are mapped to three states *active*, *_initial* and *_final* in the external TA. Further, two committed locations (©) are used to treat the two RTESM transitions *stop/stopped*. Since this external TA shows the view from the block *Timer 5*, the transition *startOrContinue/* which is originated from the environment of this block is represented as a send signal.[2] The environment variable z and the invariant on the state *active* are directly mapped to the external TA. The restart of the RTESM is specified by the transition *_reset?*.

To prove that a building block B fulfills the real-time properties stated in its own RTESM and the ones of its inner blocks, we use the transformations introduced above to create a network of TAs. It consists of the inner TA representing B's activity as well as the external TAs mapped from its RTESM and the ones of its inner blocks. In the next section, we show how this network is used to prove with UPPAAL that the real-time properties stated in the RTESMs are kept.

4 Compositional Verification of Real-Time Behavior

As mentioned in the introduction, an advantage of using ESMs to model the interface of building blocks is that the number of states to be inspected by the model checkers can be kept smaller than in monolithic proofs. This results from the fact that the ESMs represent the behavior of both the environment of a block and its inner blocks in a more abstract way. The verification of properties is provided in

[2]If we create the external TA of this RTESM from the viewpoint of the environment of *Timer 5*, e.g., block *SLS* in Fig. C2, *startOrContinue/* is modeled as a receive signal.

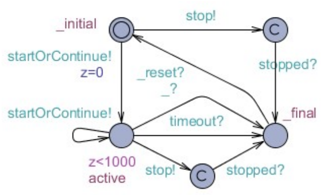


Figure C6. External block TA of building block *Timer 5*

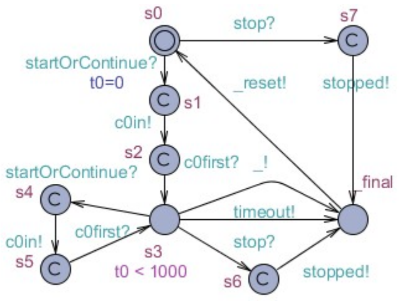


Figure C7. Internal TA of building block *Timer 5*

two separate steps (see [22, 27]). First, one verifies for each instance C of a building block in a SPACE system model that C's activity as well as the one of its environment block B fulfill C's ESM. Thereafter, if we want to prove that B meets certain properties, we can replace the activities of its inner blocks (e.g., C) with their ESMs. Since these ESMs usually contain less states than the corresponding activities as they do not model internal behavior, this reduces the state space of the model checker runs significantly.

We aim at using this kind of compositional verification also for the UPPAAL-based real-time proofs. As introduced in Sect. 3, we express real-time properties in form of states with clock invariants defined in the RTESMs. Moreover, each clock invariant is guaranteed by exactly one of the two activities to which an RTESM refers. To model which activity indeed has to guarantee a clock invariant, we use the labels of type *urgent* that are assigned to transitions leaving RTESM states with clock invariants. For instance, the clock invariant of state *active* in Fig. C3 has to be guaranteed by the block *Timer 5* since the transition */timeout*, that is labeled urgent, is triggered from this block. Below, we describe clock invariants guaranteed by the inner activity of the block as cib and the ones realized by the environment activity as cie. The verification of the two types of clock invariants is conducted by proving that the activity of a block B fulfills all clock invariants of type cib of its RTESM as well as all the clock invariants of type cie in the RTESMs of its inner blocks. As discussed in Sect. 5, these verifications are sufficient to allow reducing the proof that a system Sys fulfills a real-time property $\mathcal{P}$ to the verification that an activity $\mathcal{A}$ in Sys implies $\mathcal{P}$ as long as we represent the environment of $\mathcal{A}$ as well as its inner blocks by the respective RTESMs.

To prove that a block B indeed fulfills the mentioned clock invariants, the transformation tool uses the network of internal and external TAs described at the end of Sect. 3. It makes first a state space exploration of the combined states of this network and mark all those combined states to which one of the external TAs participate with a state containing a clock invariant that has to be guaranteed by B. For instance, to prove that the clock invariant $z < 1000$ in state *active* of the external TA of block *Timer 5* in Fig. C6 is met, we extract all combined network states in which *active* is the state component of this TA. This, are altogether five states to which the internal TA of *Timer 5* (see Fig. C7) participates with its states *s1* to *s5*.

For each state of the internal TA, that refers to a marked combined state but is not a committed location, we create a TCTL invariant stating that in this state the clock invariant of the corresponding state in the external TA is fulfilled. In our example, *s3* is the only one of the five states that is not a committed location such that we create the following invariant specified in the TCTL subset accepted by UPPAAL:

$$A[](Timer5iTA.s3\ imply\ Timer5eTA.z < 1000) \tag{1}$$

Here, the designators $Timer5iTA$ and $Timer5eTA$ refer to the internal resp. external TA of *Timer 5*.

Thereafter, we verify the created TCTL invariants with UPPAAL. It accepts formula (1) since the clock invariant of state *s3* modeling a timer node guarantees that state *s6* or *_final* will be reached within 1000 time units resulting that the external TA leaves its state *active* within this period of time as well.

It is sufficient to prove the TCTL invariants generated by our tool since all other states s of the internal TA fulfill at least one of the following two properties:

1 The state s is a committed location. In this case, it is per definition timeless and will be left without any elapsing of time. By the built-in Arctis model checker, we proved that the activity of the analyzed block is in compliance with its ESMs. This guarantees by construction that, if an internal TA awaits a signal in a committed location, the external TA sending this signal is also in a committed location such that the signal is immediately sent.

2 The state s does not participate in a combined state of the network of TAs to which one of the external TAs participates with a state carrying a clock invariant. The internal TA may rest arbitrarily long in state s since that does not violate any real-time properties.

4.1 Proving the Real-time Property of the SLS Block

Using the method described above, we can verify the second property stated in Sect. 3, i.e., that the motor may be permanently in overspeed for at most 1000 ms until the Safety Stop Emergency (SSE) component is activated. For the proof of this property, we use a network containing the internal TA of the SLS block depicted in Fig. C8 that we call i_{SLS} below. The network includes also an external TA representing the RTESM of the SLS block. It corresponds with the ESM shown in Fig. C1 but is amended with an environment clock variable z and a clock invariant $z < 1000$ stating the RTESM does not remain longer than 1000 time units (i.e., milliseconds) in the state *speedExceedSLS*. Further, the external TA of the block *Timer 5*, named eb_{T5} in the following, is part of the network. It corresponds with the TA listed in Fig. C6 but with permuted send and receive signals since, here, we reflect the view on the RTESM from the environment of block *Timer 5*. Finally, the network contains the RTESMs of the other two inner blocks *Timer Periodic* and *SLS Filter*.

For easier understanding, we colored the transitions of i_{SLS} that are synchronized with the ones of eb_{T5} in red. The state *s14* on the right upper corner of the graph in Fig. C8 indicates the state that overspeed was detected but the SSE not yet triggered. This state can only be reached from state *s2* on the top center if a *startOrContinue* signal is sent to *Timer 5*. Thus, eb_{T5} will be in state *active* which it has to leave within 1000 ms according to its clock invariant. The state *active* is left if i_{SLS} moves to state *s2* since it sends a *c1stop* (*c1* is the instance name of *Timer 5* in activity *SLS*) signal to eb_{T5}. This path models that the speed has fallen below the critical limit again. Likewise, *c1stop* is transmitted to eb_{T5} on the path to state *s12* modeling that the SLS block was disabled. Also the pass to *s22*, indicating that the SLS block was stopped, allows eb_{T5} to leave its state *active* since then the guard of the spontaneous transition to the final state is true. If not one of these three passes are followed, i_{SLS} remains in *s14* or one of the committed locations through which one circles back to *s14* when a new overspeed notification is handled. Due to the clock invariant in eb_{T5}, a time-out signal will be sent within 1000 ms, after which i_{SLS} reaches a committed location such that it immediately sends an *activateS* signal to its environment which basically guarantees the key requirement.

The states *s14* and *s28* are the only not committed states in i_{SLS} which refer to state *speedExceedSLS* in the RTESM of the SLS block. Therefore the transformation tool just created the following invariant with clock *sls* describing the external interface time constraints of the *SLS* building block:

$$A[](SLSiTA.s14||s28\ imply\ SLSeTA.sls < 1000) \tag{2}$$

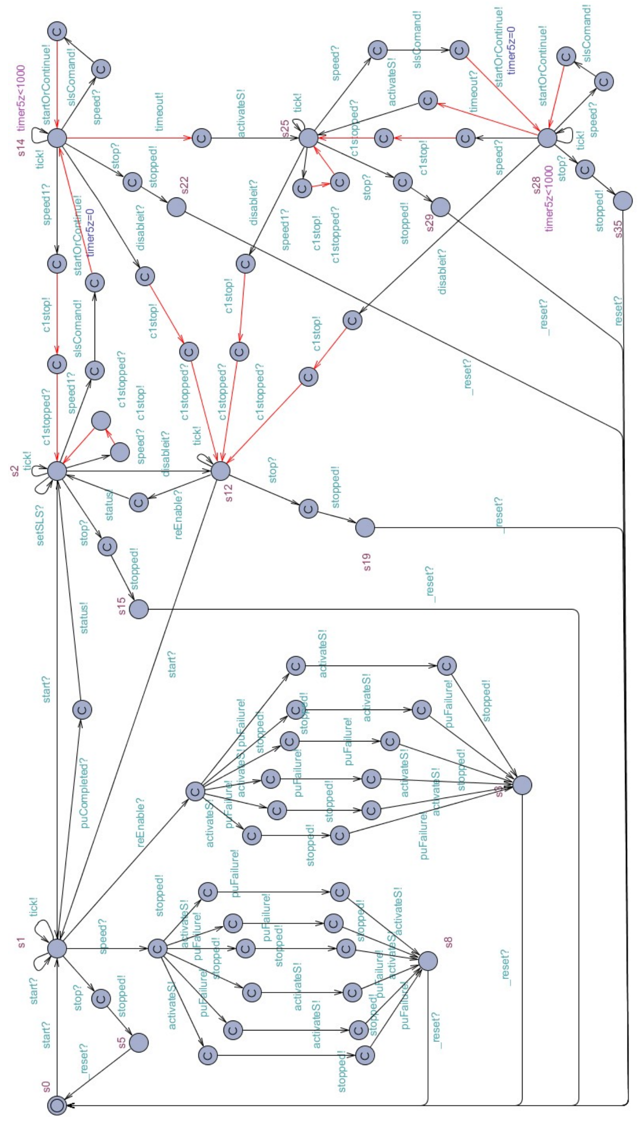


Figure C8. Internal timed automaton of SLS building block

Due to the clock invariant of the RTESM of block *Timer 5* that we proved in formula (1), UPPAAL accepted this invariant as being correct. Running on an *Windows 7* laptop system with 8 GB memory and an Intel(R) Core(TM) 2 Duo 2.40 GHz CPU, the verification of formula (2) took 0.004 second, 27352 KB of virtual memory and 7576 KB of resident memory at peak. The proof of formula (1) needed even less computing resources.

5 Correctness of the Compositional Verification

In Sect. 4, we pointed out that it is sufficient to prove certain TCTL invariant formulas with UPPAAL to verify that the activity of a block B together with real-time properties fulfilled by B's environment and its inner blocks guarantees a certain real-time property $\mathcal{P}$. To make these checks practically feasible, however, it has to be confirmed that the UPPAAL-based checks are sufficient to guarantee that $\mathcal{P}$ is also met by the system Sys containing block B. In theory, Sys might contain other building blocks which impede or delay the execution of crucial transitions leading to a violation of $\mathcal{P}$. In the following proof, we establish that our compositional concept using RTESMs as behavioral interfaces rules such real-time hampering behavior out.

A system Sys in SPACE and Arctis can be seen as a tree structure of building blocks since any inner block of an activity may contain inner blocks as well. For example, the block *Timer 5* uses a shallow block (see Sect. 2.2) from an Arctis library filtering out all but the first token reaching the block via pin *startOrContinue*. In this tree, an activity $\mathcal{A}_b$ is the parent of another activity $\mathcal{A}_c$ if $\mathcal{A}_c$ is the inner activity of block c and $\mathcal{A}_b$ its environment activity. If we want to prove that a system Sys modeled in Arctis fulfills a certain invariant real-time property $\mathcal{P}$, this corresponds to the verification of the following equation:

$$\bigwedge\nolimits_{b \in Act(Sys)} \mathcal{A}_b \Rightarrow \mathcal{P} \tag{3}$$

Here, $Act(Sys)$ refers to the set consisting of all the activities specifying system Sys.

To utilize the compositional nature of our verifications, however, we want to verify that $\mathcal{P}$ is fulfilled by a single activity $\mathcal{A}_b$ together with its RTESMs as discussed in Sect. 4. For example, the proof in Sect. 4.1 shall be sufficient to assure that the overall motor control system guarantees the real-time property stated in Sect. 3. A UPPAAL-based proof, that a real-time property $\mathcal{P}$ stated as TCTL formulas is kept by a network of TAs, corresponds to equation

$$\mathcal{A}_b \wedge ci_b \wedge \bigwedge\nolimits_{c \in Chld(b)} ci_c \Rightarrow \mathcal{P} \tag{4}$$

where ci_b denotes that the clock invariants in the RTESM of activity $\mathcal{A}_b$ are met by block b or its environment. $Chld(b)$ refers to the set of b's children in the tree mentioned above. In consequence, we have to justify that proving equation (4) is sufficient to guarantee equation (3). This can be achieved by verifying the following fomula:

$$\bigwedge\nolimits_{b \in Act(Sys)} \mathcal{A}_b \Rightarrow \forall b \in Act(Sys) : ci_b \wedge \bigwedge\nolimits_{c \in Chld(b)} ci_c \tag{5}$$

It is evident that the conjunction of equations (4) and (5) directly implies equation (3).

For the proof that equation (5) holds, we utilize the proceeding model checkers use to verify invariants. A model checker proves that the invariant is fulfilled by the initial states of a system and that none of the system transitions falsifies the invariant if it holds before. In consequence, the invariant is fulfilled by all reachable states of the system. Be S_b the set of all reachable states of activity $\mathcal{A}_b$ and $Init_{S_b} \subseteq S_b$ the set of its initial states. The SPACE approach is constraint-oriented (see [23]). That means, all state designators (i.e., queue and inner places resp. variables) are assigned to exactly one activity. Thus, we can define the system state space as the Cartesian product of all state sets in the activities (i.e., $S_{Sys} \triangleq S_1 \times \ldots \times S_n$ if $Sys = \{1, \ldots, n\}$). The set of initial system states is defined as $Init_{Sys} \triangleq \{\langle s_1, \ldots, s_n \rangle : s_i \in Init_{S_i}\}$. By the function $ls \triangleq (s : S_{Sys}, a : \{1, \ldots, n\}) \rightarrow S_a$, we map a system state s to the state component expressing the state of activity a.

Be $\widehat{T_b} \subseteq S_b \times S_b$ the set of activity steps carried out in activity $\mathcal{A}_b$. Then we define the transitions of this activity as $T_b = \widehat{T_b} \cup \{\langle s, s \rangle : s \in S_b\}$ allowing also stuttering steps in which the activity does not change its state. So, we can define the set of system transitions $T_{Sys} \triangleq T_1 \times \ldots \times T_n$ since, in a constraint-oriented model, each component takes part by either a local transition or a stuttering step in a system transition. Moreover, we use a mapping $lt \triangleq (t : T_{Sys}, a : \{1, \ldots, n\}) \rightarrow T_a$ to access the local transition of activity a carried out in the system step t. Now we can express formula (5) as the conjunct of the two following equations:

$$\begin{aligned} &\forall s \in Init_{Sys} \forall b \in Act(Sys) : \\ &\quad ci_b(ls(s, b)) \wedge \bigwedge\nolimits_{c \in Chld(b)} ci_c(ls(s, c)) \end{aligned} \tag{6}$$

$$\begin{aligned} &\forall s, s' \in S_{Sys} \forall b \in Act(Sys) : \\ &\qquad lt(\langle ls(s,b), ls(s',b)\rangle, b) \in T_b \\ &\qquad \wedge ci_b(ls(s,b)) \wedge \textstyle\bigwedge_{c \in Chld(b)} ci_c(ls(s,c)) \\ &\qquad\quad \Rightarrow ci_b(ls(s',b)) \wedge \textstyle\bigwedge_{c \in Chld(b)} ci_c(ls(s',c)) \end{aligned} \tag{7}$$

Here, $ci_b(s)$ states that the clock invariants of the RTESM of activity $\mathcal{A}_b$ hold in its state s. Equation (6) is trivially true since in the initial system state all the RTESMs are in their idle states that must not carry any clock invariants. Equation (7) is guaranteed by the TCTL invariant proofs discussed in Sect. 4. There, we proved by UPPAAL for every activity $\mathcal{A}_b$ that the clock invariants of its own RTESM as well as the ones of its inner blocks are guaranteed after carrying out any of its local transitions as long as they were preserved also before. Since $\mathcal{A}_b$ participates in a system transition either by a local activity step or a stuttering step, this implies formula (7) directly.

Thus, by combining the various proof steps discussed above, we verified that one can replace a complex UPPAAL proof using all the internal TAs of the involved Activities, i.e., equation (3), by a number of much simpler RTESM proofs, i.e., the verifications of the clock invariants fulfilling formula (7), as well as a property proof using the RTESMs of the inner blocks of an activity, i.e., equation (4). In all these proofs, a much smaller number of system states has to be checked such that the compositional verification is a useful means to circumvent the state space explosion problem.

6 Related work

Proposing model-driven development and verification of real-time systems using UML models is not a new idea. Similar to us, David et al. [24] extend UML statechart diagrams with real-time constructs and translate the resulting formalism (Hierarchical Timed Automata) into networks of TAs that can also be checked with the tool UPPAAL. In contrast to us, however, they do not utilize the structure of their models to reduce the verification overhead as we do by applying compositional verification. In [25], Knapp et al. describe their prototype tool HUGO/RT for the modeling of a generalized realroad crossing (GRC) problem. The control state machines of models are translated into Timed Automata in UPPAAL verifying the safety and utility properties of the GRC problem. In [10], Graw et al. suggest to use cTLA, a compositional extension of the Temporal Logic of Actions TLA [5], for the formal verification of UML models that describes real-time behaviors of a system. In [26], Furfaro and Nigro specify the translation of models in the formal language H-CRSM, which can also be used for the modular development of reactive real-time systems, to TAs in UPPAAL. In [27], Dong et al. summarize a series of patterns when modeling real-time systems using timed automata and provide a translation from a real time specification language, i.e., timed communication sequential process (CSP), to timed automata to facilitate the verification capabilities in UPPAAL.

Abstracting program code is another way to prove real-time constraints. For instance, Chaki et al. describe in [28] the tool MAGIC which is capable to abstract C-code into Labeled Transition Systems (LTSs) preserving the real-time properties of the code. Similarly, Gong et al. [29] construct UPPAAL models directly from source code in order to check various real-time, safety and liveness properties.

7 Conclusion and Future Work

In this paper, we presented an extension of our model-based engineering approach SPACE for reactive systems to support also the description and model checker-driven verification of real-time properties. In particular, we extended the External State Machines (ESMs) describing the interfaces of our building blocks to Real-Time ESMs (RTESMs) which enable to specify invariant real-time properties fulfilled by a building block and its environment. This approach allows for an automated transformation to Timed Automata and, in consequence, verification using the model checker UPPAAL. As shown, the approach makes it possible to use compositional verification reducing the state space to be checked significantly.

In its present state, our approach does not consider time-delays caused by executing the activity steps and, in particular, the Java methods which are supported by our current platform. In the ongoing research, we propose to take the execution time into consideration, and associate a building block with a task model. For that purpose, we can apply code level benchmarking techniques to evaluate and predicate the *best cast execution time (bcet)* and *worst case execution time (wcet)*. In the model level, we propose to translate building blocks into Timed Automata as Task models (see [31]), such that the performance and schedulability of a building block can be analyzed by such task models.

Already in its present state we consider our approach meaningful since it allows to detect real-time flaws in system designs which can already be found and corrected in the early development phase of system modeling. Due to the compositional verification, we can use the approach for real-life applications like the speed control protection mechanism for motors introduced in this paper. As missing real-time constraints are a major issue for the violation of safety properties, we consider this work as a suitable extension to our endeavor for the creation of safe embedded systems (see, e.g., [27]). The Arctis tool including standard libraries of building blocks is available from Bitreactive AS.[3]

[3]`http://www.bitreactive.com`

Bibliography

[1] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.

[2] F. A. Kraemer, "Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks," Ph.D. dissertation, Department of Telematics, Norwegian University of Science and Technology (NTNU), 2008.

[3] F. A. Kraemer, V. Slåtten, and P. Herrmann, "Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services," *Journal of Systems and Software*, vol. 82, no. 12, pp. 2068–2080, 2009.

[4] Object Management Group, "Unified Modeling Language: Superstructure, Version 2.3," 2010.

[5] F. A. Kraemer and P. Herrmann, "Reactive Semantics for Distributed UML Activities," in *Formal Techniques for Distributed Systems, Joint 12th IFIP WG 6.1 Int. Conf. (FMOODS10) and 30th IFIP WG 6.1 Int. Conf. (FORTE10)*, ser. Lecture Notes in Computer Science, J. Hatcliff and E. Zucca, Eds., vol. 6117. Springer, June 2010.

[6] ——, "Automated Encapsulation of UML Activities for Incremental Development and Verification," in *Proceedings of the 12th Int. Conference on Model Driven Engineering, Languages and Systems (MoDELS)*, ser. LNCS, A. Schürr and B. Selic, Eds., vol. 5795. Springer-Verlag, Oct. 2009, pp. 571–585.

[7] Y. Yu, P. Manolios, and L. Lamport, "Model Checking TLA+ Specifications," in *Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME99)*. London: Springer-Verlag, 1999, pp. 54–66.

[8] L. Lamport, *Specifying Systems*. Addison-Wesley, 2002.

[9] V. Slåtten, F. A. Kraemer, and P. Herrmann, "Towards Automatic Generation of Formal Specifications to Validate and Verify Reliable Distributed Systems: A Method Exemplified by an Industrial Case Study," in *Proceedings of the 10th ACM International Conference on Generative Programming and Component Engineering (GPCE11)*. ACM, 2011, pp. 147–156.

[10] J. Bengtsson, F. Larsson, P. Pettersson, W. Yi, P. Christensen, J. Jensen, P. Jensen, K. Larsen, and T. Sorensen, "UPPAAL: A Tool Suite for Validation and Verification of Real-Time Systems," 1996.

[11] R. Alur and D. Dill, "Automata for Modeling Real-Time Systems," in *Automata, Languages and Programming*, ser. Lecture Notes in Computer Science, M. Paterson, Ed. Springer Berlin / Heidelberg, 1990, vol. 443, pp. 322–335.

[12] R. Alur, C. Courcoubetis, and D. L. Dill, "Model-Checking for Real-Time Systems," in *5th Symposium on Logic in Computer Science (LICS90)*, 1990, pp. 414–425.

[13] IEC, "International Standard 61800-5-2, Adjustable Speed Electrical Power Drive Systems — Part 5-2: Safety Requirements – Functional," July 2007.

[14] L. Aceto, A. Burgueno, and K. Larsen, "Model Checking via Reachability Testing for Timed Automata," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, B. Steffen, Ed. Springer Berlin / Heidelberg, 1998, vol. 1384, pp. 263–280.

[15] L. Aceto, P. Bouyer, A. Burgueno, and K. G. Larsen, "The Power of Reachability Testing for Timed Automata," *Theoretical Computer Science*, vol. 300, pp. 411–475, May 2003.

[16] M. Lindahl, P. Pettersson, and W. Yi, "Formal Design and Analysis of a Gear Controller," in *4th International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, vol. 1384. Springer Verlag, 1998, pp. 281–297.

[17] M. Abadi and L. Lamport, "An old-fashioned recipe for real time," *ACM Transactions on Programming Languages and Systems*, vol. 16, pp. 1543–1571, 1994.

[18] G. Graw, P. Herrmann, and H. Krumm, "Verification of UML-based real-time system designs by means of cTLA," in *Proceedings of the 3rd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC2K)*. Newport Beach: IEEE Computer Society Press, 2000, pp. 86–95.

[19] N. A. Lynch and N. Shavit, "Timing-Based Mutual Exclusion," in *IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, 1992, pp. 2–11.

[20] T. A. Henzinger, "The Theory of Hybrid Automata," in *11th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1996, pp. 278–292.

[21] K. Kesten and A. Pnueli, "Timed and Hybrid Statecharts and their Textual Representation," in *Formal Techniques in Real-Time and Fault-Tolerant Systems*. Springer-Verlag, 1992, pp. 591–620.

[22] F. A. Kraemer and P. Herrmann, "Transforming Collaborative Service Specifications into Efficiently Executable State Machines," *ECEASST*, vol. 6, 2007.

[23] R. Kurki-Suonio, *A Practical Theory of Reactive Systems — Incremental Modeling of Dynamic Behaviors*. Springer-Verlag, 2005.

[24] A. David, M. O. Müller, and W. Yi, "Formal Verification of UML Statecharts with Real-Time Extensions," in *Fundamental Approaches to Software Engineering (FASE02)*, ser. Lecture Notes in Computer Science, vol. 2306. Springer-Verlag, 2002, pp. 218–232.

[25] A. Knapp, S. Merz, and C. Rauh, "Model checking - timed uml state machines and collaborations," in *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems: Co-sponsored by IFIP WG 2.2*, ser. FTRTFT '02. London, UK, UK: Springer-Verlag, 2002, pp. 395–416.

[26] A. Furfaro and L. Nigro, "Model Checking Hierarchical Communicating Real-Time State Machines," in *10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA05)*, vol. 1, Sept. 2005, pp. 6 pp. –370.

[27] J. S. Dong, P. Hao, S. Qin, J. Sun, and W. Yi, "Timed automata patterns," *Software Engineering, IEEE Transactions on*, vol. 34, no. 6, pp. 844–859, 2008.

[28] S. Chaki, E. Clarke, A. Groce, S. Jha, and H. Veith, "Modular Verification of Software Components in C," *IEEE Transactions on Software Engineering*, pp. 385–395, 2003.

[29] X. Gong, J. Ma, Q. Li, and J. Zhang, "Automatic Model Building and Verification of Embedded Software with UPPAAL," in *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE Computer Society Press, Nov. 2011, pp. 1118–1124.

[30] C. Norstrom, A. Wall, and W. Yi, "Timed Automata as Task Models for Event-Driven Systems," in *Real-Time Computing Systems and Applications, 1999. RTCSA '99. Sixth International Conference on*, 1999, pp. 182 –189.

PAPER D

Modeling Real-Time System Performance with Respect to Scheduling Analysis

Fenglin Han and Peter Herrmann

Proceedings of the 2013 International Joint Conference on Awareness Science and Technology and Ubi-Media Computing (iCAST 2013 & UMEDIA 2013). Aizu city, Japan, 2013.

MODELING REAL-TIME SYSTEM PERFORMANCE WITH RESPECT TO SCHEDULING ANALYSIS

Fenglin Han
Norwegian University of Science and Technology, Trondheim, Norway
sih@item.ntnu.no

Peter Herrmann
Norwegian University of Science and Technology, Trondheim, Norway
herrmann@item.ntnu.no

Abstract The development and analysis of embedded real-time system is complex due to its platform and application dependencies. To tackle this complexity, we amended the model-based engineering method SPACE to enable also the modeling, simulation and verification of real-time properties of reactive systems. In this paper, we present a further extension making performance estimations and schedulability analysis of reactive real-time building blocks possible. First, a performance profile for evaluating real-time tasks of a building block is outlined. Second, we present the schedulability analysis of a high level real-time system which is carried out by transforming real-time interface descriptions to timed automata that are composed with automata simulating hardware and scheduling policies.

Keywords: Real-time embedded systems, SPACE, performance evaluation, UML, timed automata.

1 Introduction

Developing reactive real-time systems is challenging since such systems have to maintain an ongoing interaction with their environment in a timely way. To achieve this, the hard real-time control of a system requires the fulfillment of stringent reliability, availability, and safety requirements. Model-based development and analysis of real-time systems (see, e.g., [1]) is considered suitable to guarantee that these requirements are kept. Further, model-based development also enables stepwise engineering with varying degrees of abstraction which facilitates a better understanding of the system properties.

SPACE is a model-based engineering method for developing reactive distributed systems. Together with its tool suite Arctis[1], it is designed in order to utilize all visual modeling, software verification and compositional system development [2, 22]. In particular, SPACE uses compositional building blocks as specification units and integrates verification and simulation techniques for developing software systems. According to our experience, up to 70% of a system model can be developed by reusing building blocks from libraries [16]. The SPACE method uses UML activity diagrams to model behavior and takes advan-

[1]Arctis is marketed by Bitreactive AS under the name Reactive Blocks, see www.bitreactive.com.

tage of so called External State-Machines (ESM) for system abstraction and system composition [16]. The activities and ESMs use a formal semantics based on the Lamport's Temporal Logic of Action (TLA, a branch of LTL) [5] (see [20]).

In [7], we extended the ESMs of the compositional building blocks to so-called Real-Time External State Machine (RTESM) with clock variables, state invariants and constraint annotations such that real-time properties, e.g, timeliness and time constraints, can be modeled and verified. The SPACE models can be translated into formal specifications in TLA [5] or TCTL [8] (see [9, 7]) such that software reliability and safety can be verified for both, functional [22] and non-functional [10] aspects. Currently, Arctis supports only the creation of Java-based systems, but extensions for C and C++ are under development.

In this paper, we first summarize the extension of SPACE to real-time embedded system design and analysis, i.e., the RTESM and its usage of compositional specification and verification of real-time systems introduced in [7]. Thereafter, we present the main contribution of this paper, i.e., a framework for component performance measurement and prediction. In particular, a set of non-functional attributes for real-time software components is defined by annotations to facilitate the non-functional analysis. Afterwards, we present a component performance measurement and schedulability analysis method.

The paper is arranged as follows: Sect. 2 summarizes the Real-Time External State Machine (RTESM) of our specification style and discusses how compositional verification is made possible. Sect. 3 presents the extension of the specifications with a performance annotation and a profiling mechanism. The performance evaluation framework is introduced in Sect. 4 while Sect. 5 shows the schedulability analysis with an example. Finally, related work is discussed in Sect. 6 followed by a conclusion in Sect. 7.

2 Real-Time External State Machines

As mentioned above, in [7] we extended the External State Machines (ESM) [16] to the Real-Time ESMs (RTESM) in order to model and verify timed properties of reactive building blocks. With annotations of selects, guards, synchronizations, updates and state invariants of environment clock variables, the RTESMs allow to express real-time properties such as limited responsiveness and time constraints. Further, the real-time properties can be automatically verified with tools such as UPPAAL [11]. Thus, like the ESMs for function properties, the RTESMs enable the compositional verification of real-time properties in SPACE.

The verification of real-time properties in our Arctis extension is prepared by automatically translating the activities and RTESMs modeling Arctis building blocks into timed automata [12] which can be recognized by UPPAAL. Then we can verify with UPPAAL whether the real-time properties are kept by a system. The automatic transformation from a UML activity-based building block to a timed automata contains the two following steps:

- **From Activity to Executable State Machine:** The activities modeling a system in SPACE are transferred to executable state machines each specifying a physical component or session [17]. The resulting framework of communicating state machines, e.g., the runtime system JavaFrame (see [14]), follows the run-to-completion semantics. In SPACE, parameter passing along control flows and actions in activity diagrams are similar to tokens traversing through enabled actions which are mapped to state machine transitions while the timers and arrival signals of an activity are translated into event triggers of the transitions.

- **From Executable State Machine to Timed State Machine:** Facilitating the extended annotations in RTESM which provide auxiliary clock variables, state invariants, updates and guards, we can enrich the expressiveness of the translated executable state machine in SPACE. Internal events such as timeouts can be constrained by auxiliary clock variables and clock invariants, such that timed requirements can be expressed and verified by corresponding tools. Parameter passing is translated to synchronization channels coordinating the synchronization of timed automata for various levels of abstraction in referenced building blocks. Decisions are modeled as compound states in executable machines that are enriched by the selects, guards, and updates of the RTESMs enabling UPPAAL to simulate each possible target state. Special transitions with global synchronization channels are added to the generated network of timed automata in order to verify the periodic simulation and verification of timed automata (see [7]).

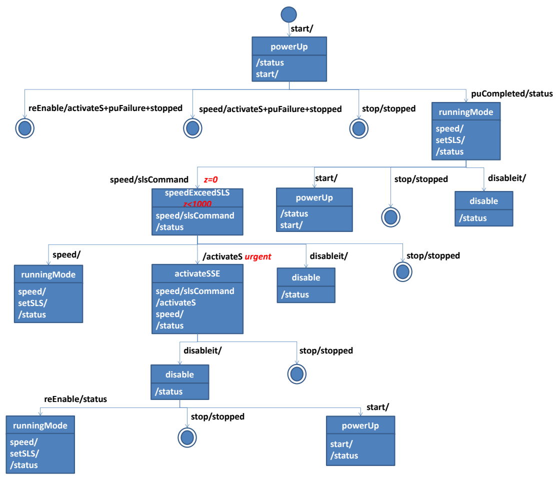


Figure D1. RTESM of Secure Limited Speed Building Block

We summarize the Real-Time External State Machine (RTESM) formalism by means of a control system for electrical motors which has been developed by Asea Brown Boveri, Ltd. (ABB). A central unit of this system is the Safety Limited Speed System component (SLS) which complies with the safety standard IEC 61800-5-2 [15] in order to guarantee that the speed of a motor always remains below a configurable maximum limit.[2] The introduction of the RTESM for the SLS component is followed by a performance annotation profile for the component model and a tentative discussion of an equivalent formalism based on petri-nets.

Figure D1 shows the RTESM of the Safety Limited Speed (SLS) component. It depicts the six different control states in the RTESM of the SLS component which are listed below:

- *idle* (expressed by the state machine starting and termination nodes): The motor control system is off.
- *powerUp*: The motor control system is starting up.
- *runningMode*: The motor is running i a normal mode not exceeding its maximum speed limit.
- *speedExceedSLS*: The motor runs above its permitted speed limit but did not exceed the maximum time period after which it has to be shut down.
- *activateSSE*: The Safe Stop Emergency (SSE) handler was triggered and the motor was shut down.
- *disable*: The SLS block is disabled after removing the power for the motor such that it cannot produce any torque again.

As Figure D1 shows, each transition contains a trigger-effect marking that describes the input and output parameters related to this action. Identifiers before the symbol / identify input parameters, e.g., *speed/* while those behind the symbol / refer to parameters leaving this block and proceeding towards the environment. Transitions with the same source and target state are listed in the boxes modeling the state. The RTESM also shows the extra labels that append timing constraints to the transitions and states. Label *z=0* shows the reset action that updates the global clock z to zero. Label $z < 1000$ shows the state invariant label presenting a time constraint on state *speedExceedSLS*. Label *urgent* defines the transition

[2]The SLS example is originally from the European founded project with the initiatives of creating Cost-Efficient methods and processes for SAfety Relevant embedded systems (CESAR, `http://www.cesarproject.eu/`).

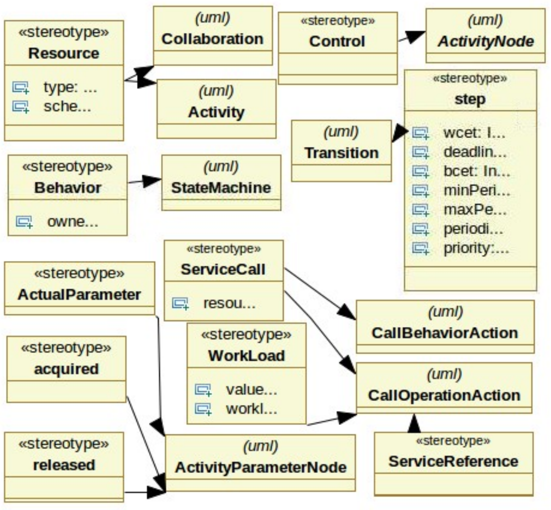


Figure D2. Building Block Performance Profile

marked */activateS* to be urgent, which means that if the time constraints on the source state is not satisfied any more. This action should be executed as soon as possible emitting an output parameter *activateS* to the environment. Various labels can be added to the model-based software component containing invariants in states, guards, actions in transitions according to the formalism of timed automata [16]. The RTESM is translated into timed automata and real-time properties under verification are expressed as Timed Computation Tree Logic (TCTL) [8] formulas. Both of them are fed to the corresponding verification tool UPPAAL [11] to verify the real-time properties in the RTESM. More detailed information about the structure of the RTESM as well as the SLS example can be found in [7].

3 Extending Component Models with Analysis Meta-Classes

In order to extend our building blocks for performance analysis, we outline the performance profile and extend the model with several annotations. In the following, we briefly introduce the extended annotation for performance analysis in our building blocks. It is inspired from the *Component Quality Model* (CQM) [17] and the *UML profile for Modeling and Analysis of Real-time Embedded System* (MARTE) [18], the OMG-standard for real-time embedded system modeling. In particular, MARTE is a hierarchical package of vocabularies and concepts provided for the communications between hardware and software developers.

Figure D2 depicts the profile used to provide extra meta-classes for annotating Arctis building blocks. Here, we view the UML elements that compose the building blocks as sub-structures of a task in a directed graph-based discrete task model. An ESM is a *UML StateMachine* composed from the elements *state* and *transition* annotated with stereotype *behavior* expressing that it is an abstract model of the interface behavior of a building block. The UML element *transition* is annotated with the stereotype *step* expressing that a transition is a step in a task model. *Collaboration* and *Activity* are annotated with the stereotype *resource* since they model the building blocks which usually are control units of software running on a resource in an embedded system. *CallOperationAction* is viewed as *ServiceCall*, which calls the underlying method written in high level language, e.g., Java or C. Each *step* in *behavior* can involve several *ServiceCalls*. The *CallBehaviorAction* call references to other building blocks which can be composed together with the current activity behavior in an event-driven manner. We listed some relevant properties of the elements in the profile and their explanations in table D1.

Some important performance statics of embedded systems, e.g., the deadline miss ratio, are typically analyzed with a set of task graphs and further petri-net models. Petri-nets are more expressive than

Table D1. Annotation Attributes

Annotation	Attributes	
	name	explanation
Step	bcet	Best case execution time
	wcet	Worst case execution time
	periodic	True if this task is periodic.
	minPeriod	Minimum interval for periodic tasks.
	maxPeriod	Maximum interval for periodic tasks.
	priority	Priority of tasks.
	ctddf	The computation time probability density distribution function.
	deadline	Maximum allowable execution time of step
Resource	type	e.g., synchronous communication channel
	schedulingPolicy	e.g., FIFO
WorkLoad	value	Integer value
	workloadType	e.g., cpu consumption
ServiceCall	ResourceType	
Behaviour	ownedStep	Contained steps
	ownedServiceCall	Contained ServiceCall
	workload	Contained workLoad

task graph models, and there is a set of extended petri-net models (e.g., stochastic activity network [19] and stochastic reward net [20]) with corresponding simulation tools. It is observable that the ESM of a building block can be divided into interleaving or sequential tasks that can have multiple instants within one instantiation of the block. Work that translates task graphs into stochastic petri-net models already exists, e.g., [21]. The translation is simple such that we do not discuss it here. The execution time of a real-time task is usually not fixed and described with a probability density distribution function like the one shown in Figure D7. Further, due to the complexity and software intensiveness of embedded systems, many numerical results are not achievable, thus only simulation results can be obtained. We propose to use the stochastic petri-net simulation tool to simulate the component performance which in our profile is expressed by the task property $ctddf$.

In our component specification, the ESM can act as a profile of internal behavior expressed by UML activities. This profiling mechanism corresponds to the rule of thumb in software development: abstraction and step-wise development. In contrast to the work in [22] which provides a direct translation from activities to petri-nets, our specification semantics is based on the tokens flows, which semantically equals to petri-net-based performance modeling and prediction methods. Figure D3 gives a petri-net model of a building block $PeriodicTimer$, which is shown in Figure D4. This timer is used in the SLS block securing a motor and issues periodically time-out signals after a certain period of time. This stochastic activity network model is equal to the ESM of building block *Periodic Timer*. The transformation mechanism will be discussed elsewhere.

4 RTESM-Based Component Evaluation

For the last several years, both researchers and industrial practitioners follow the component-based software evolution approach since software components allow to develop software systems on a more abstract and intuitive level. A survey of the application areas such as performance prediction and evaluation is presented in [23]. In this section, we introduce the RTESM-based building block evaluation using the performance profile and its annotation introduced in Sect 3.

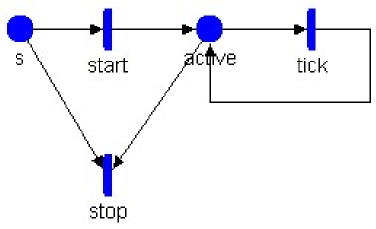


Figure D3. SAN model of building block *Periodic Timer*

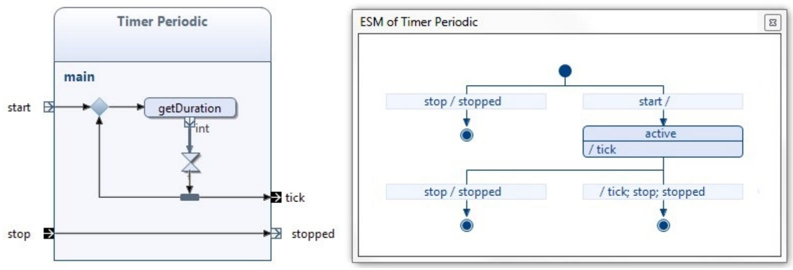


Figure D4. Building block *Periodic Timer* and its ESM

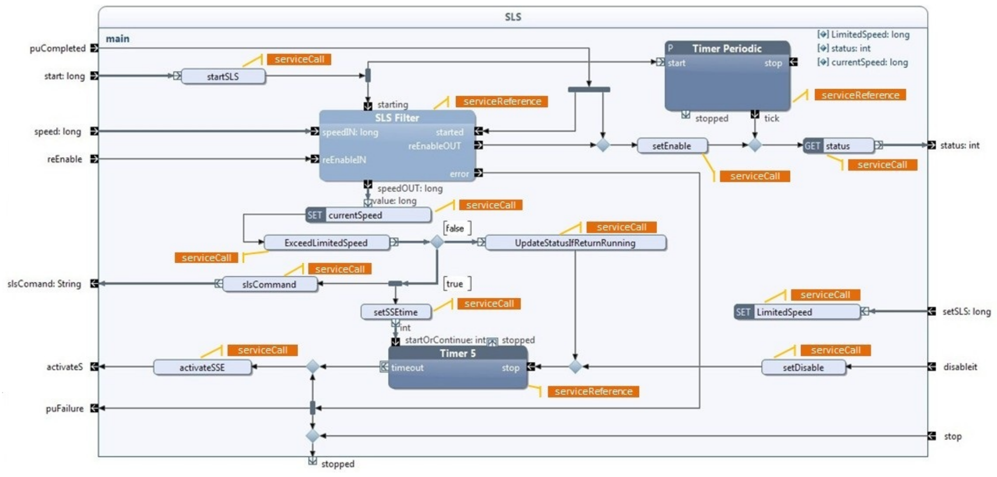


Figure D5. Secure Limited Speed Building Block with annotations

In the domain of performance analysis of real-time systems, embedded systems are considered as task and resource models. Applications are converted into tasks, and system hardware, e.g., buses and processors, are converted to resources. As previously introduced, the interface of an Arctis building block, i.e., its ESM, is a state-transition graph profile of the activity behavior which is represented by the output and input parameters of the UML activity diagram. Figure D5 is the behavioral model of the SLS component. The extended orange markings annotate the non-functional attributes of an Activity element, e.g., the *worst case execution time (wcet)* or the defined *deadline* of each method call in a *CallOperationAction*. Such abstraction-based analysis can sufficiently reduce the state space of a component model for evaluation and analysis (especially for large systems).

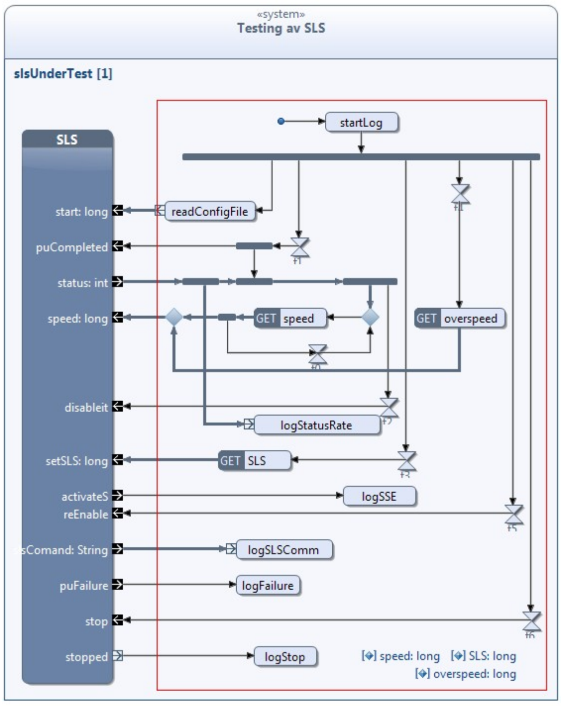


Figure D6. Monitor block for testing the response time and round-trip time for SLS

4.1 Component Utilization

Initiatively, the discrete transition time of each building block is hard to be calculated in general. The execution time of a building block *bb* is determined by the execution time of each transition and its frequency of usage. We employed the best case execution time $bcet_{bb}$ and worst case execution time $wcet_{bb}$ for component measurement. Moreover, for periodic tasks we use the inter-arrival time expressing the interval between two triggers of a task. Analyzing the utilization of the component for a given processor can give an indication of schedulability. In traditional schedulability analysis of tasks against certain processors, a utilization test is usually carried out [24]. In our approach, we apply the utilization test to the model-based software component while the component utilization is computed according to the following formula:

$$\sum_{i=1}^{n}\left(\frac{w_i}{T_i}\right) \leqslant U_{bb} \tag{1}$$

Here, w_i is the worst-case execution time of task *i* and T_i is the inter-arrival time of this task (for periodic tasks) assuming n-many tasks to be handled. The formula

$$U_{bb} = n\left(2^{\frac{1}{n}} - 1\right) \tag{2}$$

is the utilization bound for the building block *bb*. Notice that the utilization test is the lower bound of the schedulability, and the assumption is not always true, but is often used because of its simplicity.

4.2 Service Execution Time

We sketch a framework that contains a set of functions to simulate the execution time of services provided by a building block. The execution time for a block service is defined as $C = r - rt$ (see [25]),

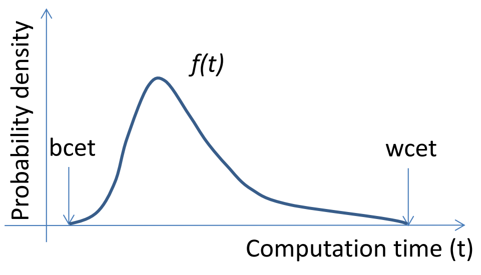


Figure D7. Task execution time distribution described by a density distribution function.

where r is the response time and rt is the round-trip time of a service. A *monitor* building block (outlined in Figure D6) is created for measuring r and rt for the SLS component. Usually, the execution time of a task in an embedded system depends on multiple factors, e.g., application, platform, environment, and thus distributes in a stochastic value between $bcet$ and $wcet$, and is expressed with a density distribution function (see Figure D7). The frequency of each transition in the ESM, i.e., the decision about the next state is provided by user data and also our simulation results. As mentioned in Sec. 3, other ongoing work is trying to convert the ESMs to stochastic petri-net models and simulate the building block execution for performance analysis. The next state decision distribution can be achieved by a simulation which may follow a steady-state distribution or be predefined. Based on the fact that a stochastic petri-net model's reachability graph can be mapped directly to a Markov process, it then satisfies the Markov property, i.e., the future states of a stochastic process depends only upon the present state, not on the sequence of events that preceded it and there is a steady-state distribution for next state choices [26].

Figure D6 shows the testing model for the *SLS* block. The red rectangle packaged area describes the behavior to monitor the *SLS* building block measuring the execution time of services provided by this building block. Initially, the log service is started via the call operation action *startLog*. The method *logStatusRate* retrieves the rate of a state information returned and calculates the time cost for retrieving such information. We define two variables *speed* and *overspeed* to simulate the types of speed value that trigger different behavior of the *SLS* block. The call operation action *logSSE* records the response time r and round trip time rt for executing *Secure Stop Emergency* (SSE). Call operation action *logSLSComm* records r and rt for action *slsCommand* reducing the speed of the motor. The rest of the operations are treated similarly. Note that the testing behavior is designed according to the state machine of the *SLS* building block, e.g., the incoming parameter $speed$ is only ready after the incoming parameter *puCompleted*. Thus, a *join* node is inserted to the control flow to enable sending the periodic $speed$ parameter. The rate of the parameter can be adjusted by clock values to test the pressure. Let S_r and S_{rt} denote two collections of measured values that contain response time and round-trip times obtained by the monitor respectively. A building block provides a set of services (denoted as s_n). A service is a path that contains several transitions in a directed graph. The worst case execution time for a service is calculated with the following formula:

$$wcet_n = max(select : \alpha_y \in S_r - select : \delta_z \in S_{rt}) \tag{3}$$

The $wcet_n$ of service n is the maximum value of the difference between too randomly selected instance measurement of S_r and S_{rt}. The $wcet$ for each transition is measured in a similar way and can be the input of the model checking-based schedulability analysis described in the following.

5 Schedulability Analysis

Below, we analyze the schedulability of building blocks using the analysis technique introduced in [27]. In contrast to the incremental verification of the real-time building blocks (see [16, 7]), the schedulability analysis is done by composing a building block with automata that simulate resources, scheduling policies, and tasks. Figure D8 shows the building block *Timer 5* which implements the emergency shut-

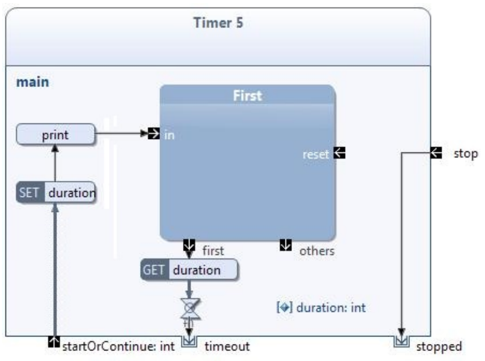


Figure D8. the Timer 5 Building Block

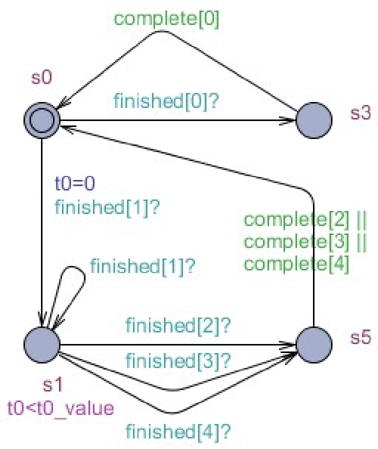


Figure D9. Translated automaton of *Timer 5* building block

down if the motor runs on overspeed for a certain period of time. The timer can be started via a periodic action which receives control signal *startOrContinue*, and it can be finished via a control path with input signal *stop* and output signal *stopped*. After receiving the control signal *startOrContinue*, the timer value is set and stored in variable *duration*. Block *First* filters out the other signals except the first one to avoid that the timer is reset by a signal *startOrContinue*. When a time out event occurs, a token is emitted out from *timeout*.

The automatically generated timed-automaton of building block *Timer 5* is shown in Figure D9. It is used as synchronization automaton for each instance of tasks and resources. Each transition with its profiled properties *wcet*, *deadline*, *periodic* is fed to the network of automata simulating tasks, resources and scheduling policies. The guard values *complete[i]* are boolean values indicating the ending of an execution period of the software component in the resource Electrical Motor Controller (EMC). The guard values *finished[i]* are semaphores used to synchronize the running tasks on the EMC and the software component automaton. The building block is automatically composed with a network of automata to form an analysis framework. All other automata can be seen in [27], i.e., the resource EMC, the task instance models of each transition in the ESM, the scheduling policies (e.g., FIFO, EDF). In the following, we mention the task instance model and how it interacts with the building block. Thereafter we will present our analysis results.

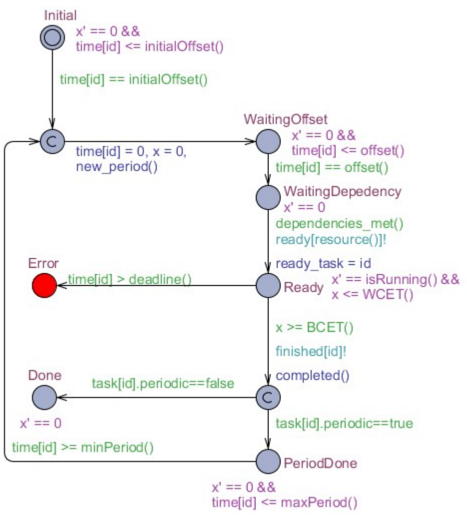


Figure D10. The Task Template from [27]

Figure D10 shows the task template for the analysis framework originating from [27]. Some changes are made to the template where the synchronization channels *finished[i]* are declared to broadcast channels since they need to synchronize both the building block automaton and the EMC automaton. The task template takes the task ID as a single parameter. The properties of a task, e.g., *wcet*, *bcet*, *deadline*, are stored in a data structure from which they can be easily retrieved. In this model, tasks are divided into periodic and non-periodic ones. After initialization from the state *Initial*, there are the following control states for a common task:

- Waiting: The waiting state is further divided into the states *WaitingOffset* and *WaitingDependency*. In *WaitingOffset* the offset time must be satisfied and *WaitingDependency* describes that tasks are waiting for certain dependencies, e.g., the precedence task and resources.
- Ready: The task is ready for execution, which means the precedence tasks are finished and the execution resource is ready to be carried out.
- Error: The *Error* simulates a state in which the task execution time, measured by clock *time[i]*, exceeds the task deadline, which causes this software component non-schedulable.
- Done: The task is finished within its deadline. A periodic task can be restarted by adding a transition from the location *periodDone* to the initiating location.

A typical resource automaton template contains the states *Idle* and *InUse* except the initiating states. The resource model is especially useful when the software is deployed to a multi-core architecture. Other automata can also be added to this network of automata, e.g., scheduling policies like DFS and FIFO as well as resources like CPU and GPU.

5.1 Verification Result

A typical problem in schedulability analysis is to check whether all tasks always meet their respective deadlines. In our simulation and verification automata, the above problem can be stated as: *Does it hold in all paths that no task is ever in the error location?* The temporal operator $\forall$ means "for all", while $\Box$ means "always", such that the following CTL formula expresses that the error state may never been reached from any of the possible system states:

$$\forall\Box forall(i : tid) not Task(i).Error \tag{4}$$

The positive verification result took 0.265 seconds and used 18848KB resident memory as well as 50688KB virtual memory with the smallest imagined deadline of each task approximately equal to two times of *wcet* (deadline=12 and wcet=6).

6 Related work

The Java Optimized Processor [28] is a Java virtual machine implementation in hardware intended for applications in embedded real-time systems. In opposite to the approach in [24] that provides an automatic translation from Java-based safety critical hard real-time systems to an abstract time preserving an UPPAAL model, we are taking a top-down approach which intends to migrate an existing matured model-based component development method to suit also the engineering of real-time systems.

Paper [29] provides a survey on Java performance evaluation approaches published in the past 10 years, and argues that more rigorous performance evaluation methodologies are needed. Moreover, statistically rigorous data analysis is advocated. Meyerhöfer and Lauterwald describe a platform independent method for component measurement. In their work in [30], platform independent component models take the Java component model running on virtual machines as an example. They are divided into basic atoms as measurement unit. In analogy, the SPACE model can be viewed as a real-world implementation of the task model. Our building block model now provides major support for Java as a script language. Our implementation also roots in the benchmarking technique applied in Java. In [31], the timed-automata formalism is attached with a task model. In particular, an extended timed automaton is viewed as an abstracted model of a running process describing the possible events that may occur during execution. TIMES is a tool-suite for schedulability analysis and synthesis of executable code for real-time systems [1]. By taking advantage of the model checking for timed automata, it can analyze task schedulability by reachability detection. Comparing the SPACE method with TIMES, we discover both methods support cyclic precedence graphs. The *wcet* and *deadline* concepts in TIMES are supported by performance annotation in the SPACE method, but SPACE is advanced in the respect that it is a complete graph model-based component model with compositional development and verification mechanism. Paper [32] gives a kernel language for component-based software performance analysis.

7 Conclusion and Future Work

In our model-based reactive real-time system development research, we translate UML activity-based specifications to timed automata in order to specify real-time properties (see also [7]). In this paper, we introduced a performance evaluation framework in which tasks associated with each transition are further clarified. It associates real-time system verification with performance evaluation and analysis. As introduced in Sect. 3, in the future we will carry out translations from our component specifications to petri-net-based performance behavior analysis, especially using the stochastic petri-net oriented analysis models and tools. In nature, the building block component model is very similar to any control systems that reveals the logic of functional behavior and abstracts away code details such as condition and loop control.

Often, an entire system cannot be analyzed due to high costs or since some required services cannot be provided (see [33]). Thus, performance evaluation and prediction is an important but complex task in the development of new software and hardware system development. Our purpose is to provide an integrated environment for model-based real-time system development, verification, analysis and simulation.

In the future, we want to integrate a real-time language into our code generation platform, e.g., the real-time Java profile developed in [28]. Furthermore, low level benchmarking techniques afford the integration into our model-based system development platform, e.g., the Integer Linear Programming technique in [34]. Such a translation will be provided for translating building blocks into the Stochastic Activity Network (SAN) for performance prediction and simulation.

Acknowledgements

This work is partially funded by the Research Council of Norway, under the research and development project “Infrastructure for Integrated Services” (ISIS).

Bibliography

[1] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "TIMES: A Tool for Schedulability Analysis and Code Generation of Real-Time Systems," in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science, K. Larsen and P. Niebert, Eds. Springer-Verlag, 2004, vol. 2791, pp. 60–72. [Online]. Available: `http://dx.doi.org/10.1007/978-3-540-40903-8_6`

[2] F. A. Kraemer, R. Bræk, and P. Herrmann, "Compositional Service Engineering with Arctis," *Telektronikk*, vol. 105, no. 2009.1, 2009.

[3] F. A. Kraemer, V. Slåtten, and P. Herrmann, "Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services," *Journal of Systems and Software*, vol. 82, no. 12, pp. 2068–2080, December 2009.

[4] F. A. Kraemer and P. Herrmann, "Automated Encapsulation of UML Activities for Incremental Development and Verification," in *Proceedings of the 12th Int. Conference on Model Driven Engineering, Languages and Systems (MoDELS)*, ser. LNCS, A. Schürr and B. Selic, Eds., vol. 5795. Springer-Verlag, Oct. 2009, pp. 571–585.

[5] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[6] F. A. Kraemer and P. Herrmann, "Reactive semantics for distributed uml activities," in *Formal Techniques for Distributed Systems*, 2010, pp. 17–31.

[7] F. Han, P. Herrmann, and H. Le, "Modeling and Verifying Real-time Properties of Reactive Systems," in *18th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS2013)*, 2013.

[8] R. Alur, C. Courcoubetis, and D. L. Dill, "Model-Checking for Real-Time Systems," in *5th Symposium on Logic in Computer Science (LICS90)*, 1990, pp. 414–425.

[9] F. A. Kraemer, V. Slåtten, and P. Herrmann, "Engineering Support for UML Activities by Automated Model-Checking — An Example," in *Proceedings of the 4th International Workshop on Rapid Integration of Software Engineering Techniques (RISE)*, November 2007.

[10] G. Graw, P. Herrmann, and H. Krumm, "Verification of UML-based real-time system designs by means of cTLA," in *Proceedings of the 3rd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC2K)*. Newport Beach: IEEE Computer Society Press, 2000, pp. 86–95.

[11] J. Bengtsson, F. Larsson, P. Pettersson, W. Yi, P. Christensen, J. Jensen, P. Jensen, K. Larsen, and T. Sorensen, "UPPAAL: A Tool Suite for Validation and Verification of Real-Time Systems," 1996.

[12] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[13] F. A. Kraemer and P. Herrmann, "Transforming Collaborative Service Specifications into Efficiently Executable State Machines," in *Proceedings of the 6th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007)*, ser. Electronic Communications of the EASST, K. Ehring and H. Giese, Eds., vol. 7. EASST, 2007.

[14] Ø. Haugen and B. Møller-Pedersen, "B.: JavaFrame — Framework for Java Enabled Modelling," in *In: Proc. Ericsson Conference on Software Engineering*, 2000.

[15] IEC, "International Standard 61800-5-2, Adjustable Speed Electrical Power Drive Systems — Part 5-2: Safety Requirements – Functional," July 2007.

[16] L. Aceto, A. Burgueno, and K. Larsen, "Model Checking via Reachability Testing for Timed Automata," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, B. Steffen, Ed. Springer Berlin / Heidelberg, 1998, vol. 1384, pp. 263–280.

[17] V. Grassi, R. Mirandola, E. Randazzo, and A. Sabetta, "Klaper: An intermediate language for model-driven predictive analysis of performance and reliability," in *The Common Component Modeling Example: Comparing Software Component Models [result from the Dagstuhl research seminar for CoCoME, August 1-3, 2007]*, ser. Lecture Notes in Computer Science, A. Rausch, R. Reussner, R. Mirandola, and F. Plasil, Eds., vol. 5153. Springer, 2007, pp. 327–356.

[18] "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems." [Online]. Available: `http://www.omg.org/omgmarte/Specification.htm`

[19] A. Bidgoly, A. Khalili, and M. Azgomi, "Implementation of coloured stochastic activity networks within the pdetool framework," in *Modelling Simulation, 2009. AMS '09. Third Asia International Conference on*, 2009, pp. 710–715.

[20] C. Constazltinescu and T. Trivedi, "A stochastic reward net model for dependability analysis of real-time computing systems," in *Real-Time Applications, 1994., Proceedings of the IEEE Workshop on*, 1994, pp. 142–146.

[21] A. R. McSpadden and N. Lopez-Benitez, "Stochastic petri nets applied to the performance evaluation of static task allocations in heterogeneous computing environments," in *Proceedings of the 6th Heterogeneous Computing Workshop (HCW '97)*, ser. HCW '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 185–. [Online]. Available: `http://dl.acm.org/citation.cfm?id=795688.797847`

[22] J. P. López-Grao, J. Merseguer, and J. Campos, "From UML activity diagrams to Stochastic Petri nets: application to software performance engineering," in *Proceedings of the 4th international workshop on Software and performance*, ser. WOSP '04. New York, NY, USA: ACM, 2004, pp. 25–36. [Online]. Available: `http://doi.acm.org/10.1145/974044.974048`

[23] A. Alvaro, E. de Almeida, and S. Meira, "A software component quality model: A preliminary evaluation," in *Software Engineering and Advanced Applications, 2006. SEAA '06. 32nd EUROMICRO Conference on*, 29 2006-sept. 1 2006, pp. 28 –37.

[24] T. Bøgholm, H. Kragh-Hansen, P. Olsen, B. Thomsen, and K. G. Larsen, "Model-based schedulability analysis of safety critical hard real-time Java programs," in *Proceedings of the 6th international workshop on Java technologies for real-time and embedded systems*, ser. JTRES '08. New York, NY, USA: ACM, 2008, pp. 106–114. [Online]. Available: `http://doi.acm.org/10.1145/1434790.1434807`

[25] R. Perrone, R. Macedo, G. Lima, and V. Lima, "An approach for estimating execution time probability distributions of component-based real-time systems," vol. 15, no. 11, pp. 2142–2165, jun 2009, `http://www.jucs.org/jucs_15_11/an_approach_for_estimating`.

[26] W. Feller, *An Introduction to Probability Theory and Its Applications*. Wiley, January 1968, vol. 1. [Online]. Available: `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20{&}path=ASIN/0471257087`

[27] A. David, J. Illum, K. G. Larsen, and A. Skou, *Model-Based Design for Embedded Systems*. CRC Press, 2010, ch. Model-Based Framework for Schedulability Analysis Using UPPAAL 4.1, pp. 93–119.

[28] M. Schoeberl, "JOP: A Java Optimized Processor for Embedded Real-Time Systems," Ph.D. dissertation, Vienna University of Technology, 2005. [Online]. Available: `http://www.jopdesign.com/thesis/thesis.pdf`

[29] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous java performance evaluation," in *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, ser. OOPSLA '07. New York, NY, USA: ACM, 2007, pp. 57–76. [Online]. Available: `http://doi.acm.org/10.1145/1297027.1297033`

[30] M. Meyerhöfer and F. Lauterwald, "Towards platform-independent component measurement," in *in Tenth International Workshop on Component-Oriented Programming*, 2005.

[31] C. Norstrom, A. Wall, and W. Yi, "Timed automata as task models for event-driven systems," in *Real-Time Computing Systems and Applications, 1999. RTCSA '99. Sixth International Conference on*, 1999, pp. 182 –189.

[32] V. Grassi, R. Mirandola, and A. Sabetta, "From design to analysis models: a kernel language for performance and reliability analysis of component-based systems," in *Proceedings of the 5th international workshop on Software and performance*, ser. WOSP '05. New York, NY, USA: ACM, 2005, pp. 25–36.

[33] M. Kuperberg and S. Becker, "Predicting software component performance: On the relevance of parameters for benchmarking bytecode and apis," in *Proceedings of the 12th International Workshop on Component Oriented Programming (WCOP 2007)*, 2007.

[34] M. Schoeberl and R. Pedersen, "WCET analysis for a Java processor," in *Proceedings of the 4th international workshop on Java technologies for real-time and embedded systems*, ser. JTRES '06. New York, NY, USA: ACM, 2006, pp. 202–211. [Online]. Available: `http://doi.acm.org/10.1145/1167999.1168033`

PAPER E

Towards Verifying Safety Properties of Real-Time Probabilistic Systems

Fenglin Han, Jan Olaf Blech, Peter Herrmann and Heinz Schmidt

Proceedings of the 11th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA 2014), Electronic Proceedings in Theoretical Computer Science 147, pp. 1-15. Grenoble, France. April 12th, 2014.

TOWARDS VERIFYING SAFETY PROPERTIES OF REAL-TIME PROBABILISTIC SYSTEMS

Fenglin Han
Norwegian University of Science and Technology,
Trondheim, Norway
sih@item.ntnu.no

Jan Olaf Blech
RMIT University, Melbourne, Australia
janolaf.blech@rmit.edu.au

Peter Herrmann
Norwegian University of Science and Technology,
Trondheim, Norway
herrmann@item.ntnu.no

Heinz Schmidt
RMIT University, Melbourne, Australia
heinz.schmidt@rmit.edu.au

Abstract Using probabilities in the formal-methods-based development of safety-critical software has quickened interests in academia and industry. We address this area by our model-driven engineering method for reactive systems SPACE and its tool-set Reactive Blocks that provide an extension to support the modeling and verification of real-time behaviors. The approach facilitates the composition of system models from reusable building blocks as well as the verification of functional and real-time properties and the automatic generation of Java code.

In this paper, we describe the extension of the tool-set to enable the modeling and verification of probabilistic real-time system behavior with the focus on spatial properties that ensure system safety. In particular, we incorporate descriptions of probabilistic behavior into our Reactive Blocks models and integrate the model checker PRISM which allows to verify that a real-time system satisfies certain safety properties with a given probability. Moreover, we consider the spatial implication of probabilistic system specifications by integrating the spatial verification tool BeSpaceD and give an automatic approach to translate system specifications to the input languages of PRISM and BeSpaceD. The approach is highlighted by an example.

1 Introduction

Modeling and verification methods for embedded control system in domains such as avionics, automotive and robotics should address a variety of software and hardware aspects including real-time and probabilistic properties, distribution of system components, communication protocols, characteristics of digital circuits and controllers. Real-time systems can require quantitative timing constraints which may include guaranteed probabilities for time and spatial properties. For example, a robot may be required to process a task in a predefined amount of time with a probability of 99.999999% to prevent expensive maintenance operations resulting from minor damage to the equipment. It must complete the task in a slightly larger amount of time with 100% to prevent major damage.

Here, we propose a framework for integrating probabilistic real-time verification and performance prediction with system development. This approach extends our existing model-driven engineering framework SPACE and its tool-set Reactive Blocks[1] [22] with real-time system behavior verification and schedulability analysis [11, 12]. Reactive Blocks enables the model-based engineering of reactive systems by composing reusable building blocks each describing a certain sub-functionality of a system. The composed system model is automatically transformed into executable Java code [22]. Further, various formal verification methods ensure functional correctness [22] as well as reliability [27], security [9] and safety [11, 12] of targeted systems.

The formalism for modeling probabilistic real-time systems used in this work is based on probabilistic timed automata (PTA) [22]. It incorporates both probabilistic and real-time characteristics. Probabilistic properties are represented with an extension of Computational Tree Logic, i.e., PCTL [14]. PRISM [23] is a probabilistic model checker for formal analysis of systems that exhibits stochastic behaviors. It supports multiple-formalisms, including discrete-time Markov chains, continuous-time Markov chains, Markov decision processes and PTA making it possible to capture the random behavior of our real-time system model [11], for example random aspects of failure or uncertain inputs, loads or timing. We choose PTA for the stochastic behavior since it integrates well with our existing timed-automata based real-time system model. PTA has an equivalent descriptive power to a MDP (Markov Decision Process) [15] such that we can use the terminology of MDP for the system descriptions. Probabilistic CTL [18] has the capability of expressing real-time as well as probabilistic properties of a reactive system.

The tool set described in this paper involves the five engineering steps outlined below:

1 We model a system using Reactive Blocks including a simulator of its environment, in particular the spatial conditions to be reflected. In this model, we annotate probabilities as well as real-time behavior.

2 The model is analyzed with the model checker built into Reactive Blocks for functional errors and transformed into an executable simulator.

3 The simulator is carried out and, during the simulation, traces capturing spatiotemporal behavior with or without annotated probabilities are extracted for spatial verification.

4 The extracted spatiotemporal behavior is verified for possible spatial implications like collisions using our BeSpaceD tool [4]. Here, distributions capturing the combined probabilistic time behavior of the subcomponents are created from the extended Reactive Block models using the PRISM-based analysis.

5 If all analyses are passed, the simulator sub-functionality is removed from the Reactive Blocks model such that its core functionality can be transformed into executable code.

In this paper, we have three main contributions.

1 A novel approach for system performance predictions is introduced. In particular, we present a probabilistic real-time state-machine for software component performance descriptions. This so-called PRTESM is an extension of the External State Machines (ESMs) [16] used in Reactive Blocks. It allows to express probabilistic real-time assumptions and guarantees of a building block. That enables us to compose the PRTESMs of the various building blocks forming a system to predict probabilities of the overall system behavior.

2 We show the integration of the model-checker PRISM [23] to the tool-set. For that, the PRTESMs are transformed into PTAs [22] and the performance predictions of the overall system to PCTL statements [14] which can be directly proven by PRISM.

[1]Until recently, Reactive Blocks was called *Arctis*.

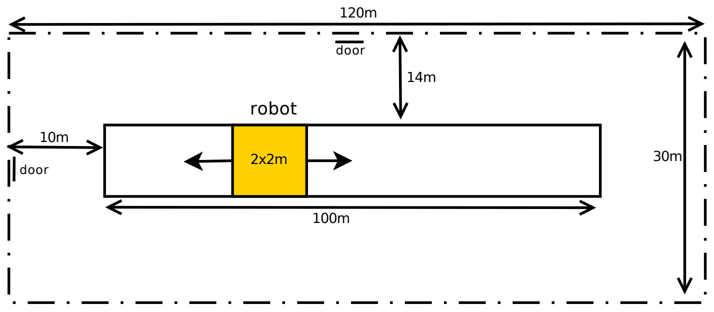


Figure E1. Layout of the moving robot

3 We look at spatial implications of probabilities in system behavior. The introduction of known probabilities into system behavior allows us to calculate how likely a physical unit like a robot will be present in a given area in space. We use the tool BeSpaceD [4] for this.

1.1 Guiding Example

We illustrate the tool set by a scenario of a moving robot in a factory hall featuring probabilistic behavior. Figure E1 provides a spatial layout of the example scenario. The moving robot occupies a 2 x 2 meters space in the 120 x 30 meters factory hall and moves along a straight line in the center of the room covering a distance of 100 meters. The maximum speed of the robot can reach 10 $\frac{m}{s}$ and thus a collision with a human may lead to fatal injuries. To eliminate such injuries, the hall is equipped with sensors monitoring the robot for approximations of humans. If the robot comes close to a human, it is slowed down or even stopped. The probabilistic aspect of this example comprises probability distributions on the reaction time once a human is detected. In this paper, the robot controller and a simulator of the continuous robot behavior are developed and implemented using Reactive Blocks.

1.2 Overview

The paper is arranged as follows: In Section 2, we give a description of the model of the robot control system example in Reactive Blocks realizing the distributed control functions as well as the robot simulation. Section 3 introduces our formalism for probabilistic time constraint and the translation into PRISM input. Section 4 presents our approach for spatial implications of probabilistic system behaviors and introduces the tool for probabilistic spatiotemporal property verification. We present the verification of our properties in Section 5. Related work is discussed in Section 6 followed by a conclusion in Section 7.

2 Modeling Control Functions

In the moving robot example sketched in Section 1.1, the three main activities are:

1 Polling of sensor data about the positions of the robot and the human[2].

2 Deciding the correct operation mode of the robot based on the distance to the human.

3 Forwarding an altered operation mode to the robot controller.

All three activities together should be performed within $0.5\ s$ at maximum.

We model these control functions as well as the simulator of the human and robot behaviors separately from each other by different building blocks. In SPACE and Reactive Blocks, a building block consists

[2]For simplicity, we only consider the human closest to the robot.

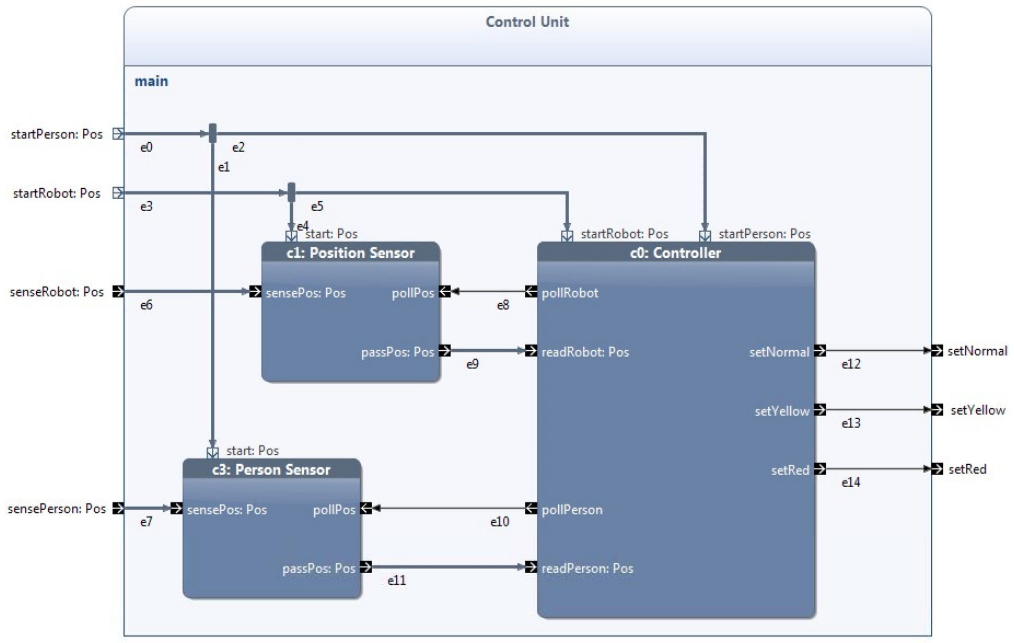


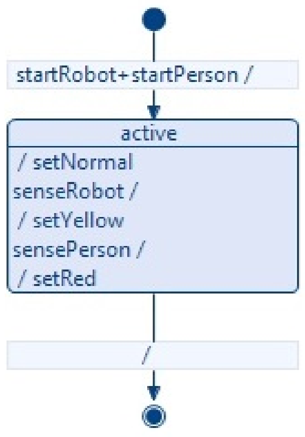
Figure E2. UML activity of building block *Control Unit*

of a behavioral model in the form of a UML activity [22] supplemented by an External State Machine (ESM) [16] describing its interface behavior.

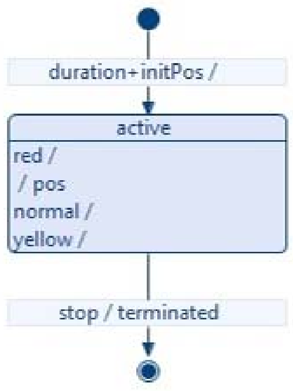
The safety controller protecting humans from collisions with the fast moving robot is specified by the building block *Control Unit*. The UML activity modeling the behavior of this block is shown in Figure E2. Similar to Petri Nets, the control and data flows are represented by the flow of tokens in the activities. These tokens are passed by the activity edges towards vertices. Vertices can be control elements (such as forks duplicating tokens) or operations (associated with Java methods executed when a token arrives). Further, activities may contain call behavior actions like *Controller*, *Position Sensor* and *Person Sensor* each referring to another building block. The interaction between the activity containing a call behavior action of a building block and the one referring to its behavior is modeled by pins and parameter nodes. Parameter nodes are the identifiers at the edge of an activity, e.g., *startRobot* in block *Control Unit*. All parameter nodes of an activity are available as pins in the call behavior actions referring to its building block (e.g., *setRed* in block *Controller*). The semantics defines that a token reaching a pin of a call behavior action continues from the corresponding parameter node of the activity referring to the behavior of the call behavior action and vice versa.

Position Sensor and *Person Sensor* refer to the sensors for the positions of the robot and human which get the current position information from the simulation via their input pins *sensePos*. The block *Controller* realizes the safety controller of the system. It polls the robot and human positions every 10 ms using the pins *poll* and *read*. From these inputs, the distance between human and robot is computed and the correct operation mode is selected. Altogether, there are three operation modes that are defined as follows:

- *Normal mode:* If no human is closer than 25 meters to the robot, the robot accelerates with $5\frac{m}{s^2}$ until reaching a speed of $10\frac{m}{s}$. When it is $11\ m$ close to its endpoint, it decelerates with $5\frac{m}{s^2}$ until reaching a speed of $1\frac{m}{s}$ which is carried until reaching the endpoint.



(a) ESM of *Control Unit*.



(b) ESM of *Robot Operation*.

Figure E3. ESMs of building blocks *Control Unit* and *Robot Operation*

- *Yellow mode:* If a human is detected in a distance of less than 25 meters but more than 10 meters, the robot is slowed down with a rate of $10\frac{m}{s^2}$ until reaching a speed of $2\frac{m}{s}$ (resp. $1\frac{m}{s}$ if it is closer than $11\ m$ to its endpoint).
- *Red mode:* If the human is within 10 meters range to the robot, the robot makes an emergency stop with a deceleration of $15\frac{m}{s^2}$.

The behavior of the building block *Control Unit* is specified by its ESM depicted in Fig. 12.3(a). An ESM is a UML state machine describing which of its parameter nodes are passed by tokens in a certain transition. *Control Unit* is initiated by parallel flows through the parameter nodes *startRobot* and *startPerson* which contain the initial positions of robot and human. Thereafter, the building block is in state *active* in which the environment (symbol $/$ right of the parameter node identifier) may send position data via parameter nodes *senseRobot* and *sensePerson* while from the block itself ($/$ left of the parameter node identifier) the current operation mode may be sent via *setNormal*, *setYellow* and *setRed*. The building block is terminated and all remaining tokens on its activity are removed if the activity containing the call behavior action of the block is terminated as well which is described by the transition $/$.

Figure E4 shows the UML activity of building block *Robot Operation*. It contains the building block *Mode Selector* storing the current operation mode. *Robot Controller* models the controller of the robot, which chooses the current robot speed according to the operation mode and the position of the robot in the factory hall. Using the block *Robot Physics* we model the simulator for the robot. The continuous behavior is specified by a difference equation which is executed every $5\ ms$.

The ESM of block *Robot Operation* in Figure 12.3(b) determines that the block is started by parallel token flows via the parameter nodes *duration* and *initPos* which refer to the execution time of the difference equation (i.e., $5\ ms$) and the initial position of the robot. In state *active*, the operation modes *normal*, *yellow* or *red* may be received by the environment while by *pos* the current position of the robot may be forwarded towards the sensor in block *Control Unit*. The block is terminated by a token coming via parameter node *stop* which leads to an output via *stopped*. This signal leads to the termination of all inner blocks followed by the termination of *Robot Operation*.

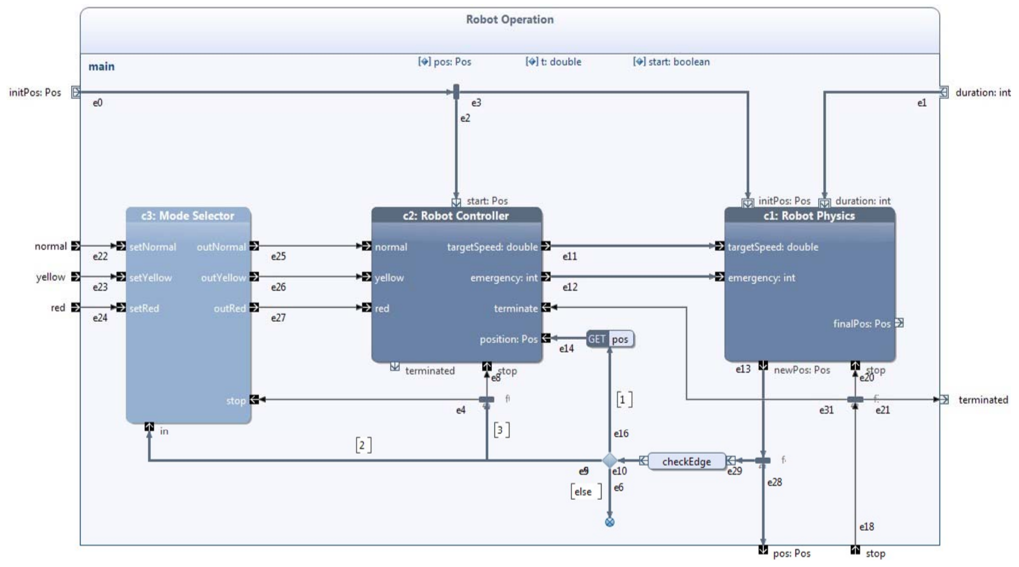


Figure E4. UML activity of building block *Robot Operation*

The third block of the example is called *Suicidal Human*. This somehow odd name refers to the simulation of a human attempting to approach the robot as fast as humanly possible. So, it describes the worst case situation to be solved by the safety controller. Like the robot, the behavior of the human is specified by a difference equation that is executed every $5\ ms$. For the sake of brevity, we do neither list this block nor the other blocks of our system in detail here.

After creating and composing all building blocks of our system, we can check them for the presence of functional design errors like not fulfilling their ESMs (see [22]). Further, if all checks are passed, Reactive Blocks automatically generates executable Java of our system which can be carried out to simulate robot runs.

2.1 Probability Assumptions and Temporal Safety

As discussed above, the factory hall, in which humans and machines collaborate in close proximity, is monitored by camera sensors for collision avoidence. A safety controller constantly monitors the operation for the proximity of humans and then decides which operation mode to choose. Of course, to avoid collisions when the robot is still moving, we have to guarantee maximum reaction times for the different functions carried out in order to slow down or stop the robot. The four main sub-tasks are the fetching of sensor data including the delay between two pollings of the sensors, the processing time of the safety controller in order to compute the distances between human and robot and to decide about the correct operation mode, the communication delay between the safety and the robot controller as well as the processing time of the robot controller including delays within the robot starting to break.

For elaborating our approach, we assume that the probabilities for the reaction times associated with the sub-tasks correspond to the percentages depicted in Table F1. They show the probability that a task is finished within a certain time in an accumulative way. For instance, according to the table, the fetching of the sensor data is finished within $15\ ms$ with a probability of $10\ \%$ while the overall probability that it is completed within $17\ ms$ is $40\ \%$. Thus, we guarantee that this task is carried out within $20\ ms$. Of course, in practice one cannot give axiomatic guarantees of real-time properties since a control system is always subject to external influences like a failure of the computer hardware running it. We decided to ignore these kinds of external error in our models but are aware that, when our tool chain is used for

Table E1. Accumulative probability distribution of the execution times for the different tasks

Delay Type	Maximum Time	Probability	Fig. E5	Fig. E6
Time to fetch sensor data including polling delay	15 ms	10 %		
	17 ms	40 %		
	18 ms	85 %		
	19 ms	99.998 %		
	20 ms	100 %		
Processing time recognition unit	250 ms	10 %		
	260 ms	30 %		
	270 ms	60 %		
	280 ms	90 %		
	285 ms	99 %		
	290 ms	100 %		
Communication time recognition unit to robot	15 ms	80 %	r1	
	16 ms	98 %	r2	
	16.5 ms	99.5 %	r3	
	16.9 ms	99.99999995 %	r4	
	20 ms	100 %	r5	
Internal robot processing time and actuator reaction	150 ms	5 %		r1
	159 ms	90 %		r2
	160 ms	95 %		r3
	165 ms	99.9995 %		r4
	170 ms	100 %		r5

real hazard analysis, such faults have to be taken into account as well. The values in the table do not correspond to an existing system, but rather represent typical values one might expect in some field-bus based systems.

3 Probabilistic Real-Time Extended State Machines

Following the concept of Timed Automata [1], we extended our external state machines (ESM) to Real-Time ESMs (RTESM) in [11, 12]. RTESMs allow the specification of deadlines for the time a building block may rest in a certain RTESM state. As a new contribution in this paper, we introduce the further extension of the RTESMs to Probabilistic Real-Time External State Machines (PRTESM). Extending the habitual pattern in Reactive Blocks to model functional and non-functional interface properties of building blocks, the PRTESMs make the description of probabilistic real-time behavior possible and allow to describe discrete probability distributions like the ones listed in Table F1. PRTESMs allow a straightforward transformation into Probabilistic Timed Automata (PTA) [22] that can be model checked by verification tools like PRISM [23, 20].

Figures E5 and E6 show the PRTESMs of the blocks *Control Unit* and *Robot Operation*. To facilitate the transformation into PTAs, a PRTESM contains an initial state *initial* representing both the initial and final states of the corresponding ESM. Moreover, the PRTESM may contain special states that express probabilistic behavior of the concerned actions as well as the synchronization semaphores and timed constraints used to model real-time properties. In the PRTESMs listed in Figures E5 and E6, the values **r1**, **r2**, **r3**, **r4**, **r5** represent the probabilities from the third resp. forth section in Table F1. The time deadlines are measured in 100 microseconds.

The approach for the generation of a PTA from a PRTESM for a real-time blocks is semi-automatic.

- First, a set of communication actions are identified in the building block concerning the underneath hardware or communication protocol. In our example in the building block *Control Unit*, the parameter pins *setNormal, setYellow, setRed* and *sensePerson* realize the communication among distributed agents in the moving robot scenario. The *setX* set of parameters realize communication

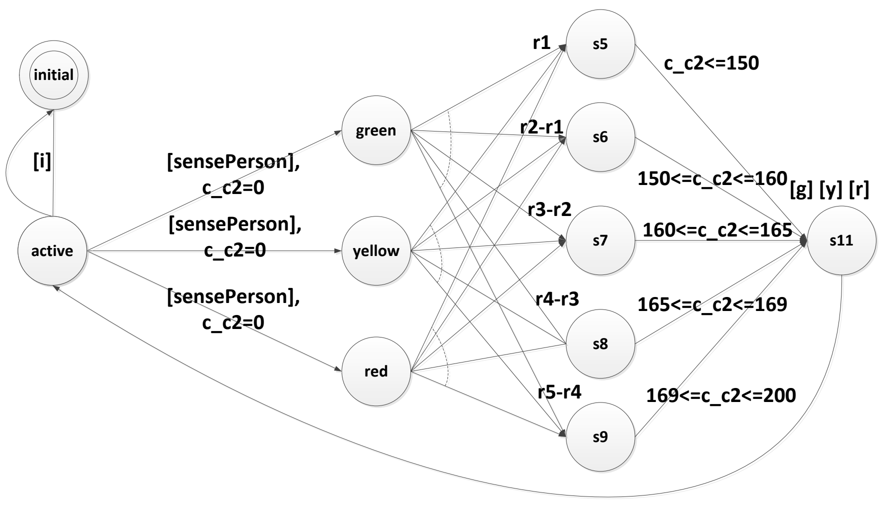


Figure E5. PRTESM for the $ControlUnit$ block

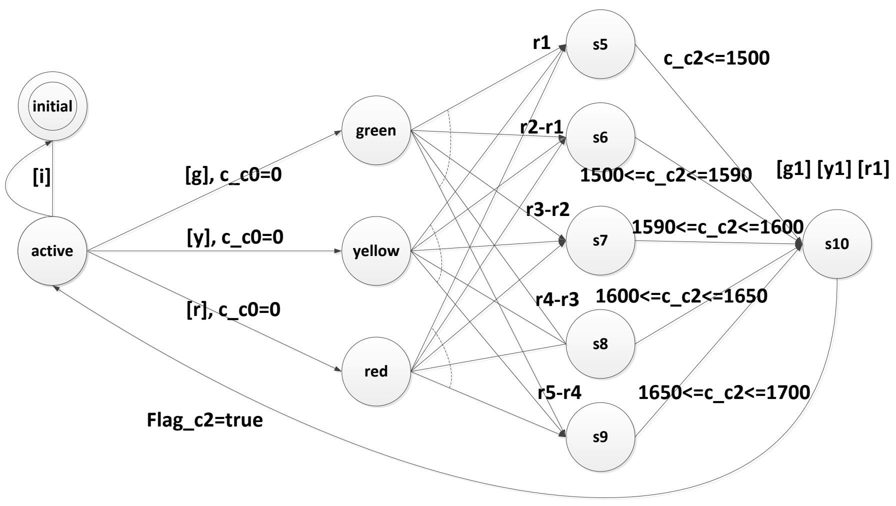


Figure E6. PRTESM for $RobotOperation$ block

between robot controller and robot actuator while the *sensePerson* parameter realizes the communication between camera sensor and robot controller. Thus we extract the probabilistic real-time actions in the system to PRTESM and ignore other actions.

- Second, transitions corresponding to distributed communications are transformed to new states and transitions expressing probabilities. In our example these are: $active \rightarrow_{setNormal} active$, $active \rightarrow_{setYellow} active$, $active \rightarrow_{setRed} active$ and $active \rightarrow_{sensePerson} active$.

The code excerpt in Figure E7 corresponds to the PRTESM in Figure E5 and illustrates the corresponding Probabilistic Timed Automata (PTA) of the building block *Control Unit*. The formalism declaration *pta* demands that the following modules follow the timed automata style. Time and probability values are declared as constant values before the module declaration. *c2_Control_Unit_prtesm*

```
1. pta
2.  const int c2_1 = 150; // time unit 0.0001 s
3.  const int c2_2 = 160;
4.  ...
5. const double r1 = 0.8;  // probability
6. const double r2 = 0.98; // accumulative probability
7. module c2_Control_Unit_prtesm
8.        s_c2 : [0..10] init 0;
9.        c_c2 : clock;
10.       flag_c2 : bool init false;
11.       [i] s_c2=0 -> (s_c2'=1);
12.       [r] s_c2=1 -> (s_c2'=2)&(c_c2'=0);
13.       ...
14.       [sensePerson] s_c2=1 -> (s_c2'=2)&(c_c2'=0);
15.       [] s_c2=2 -> r1 : (s_c2'=5) + r2-r1 : (s_c2'=6)
16.       + r3-r2 : (s_c2'=7) + r4-r3 : (s_c2'=8)
17.       + r5-r4 : (s_c2'=9);
18.       [y] s_c2=5&c_c2<=c2_1 -> (s_c2'=10);
19.       [y] s_c2=6&c_c2>=c2_1&c_c2<=c2_2 -> (s_c2'=10);
19.       ...
20.       [] s_c2=10 -> (s_c2'=10) & (flag_c2'=true);
21.endmodule
```

Figure E7. Excerpt of PTA codes corresponds to Figure E5.

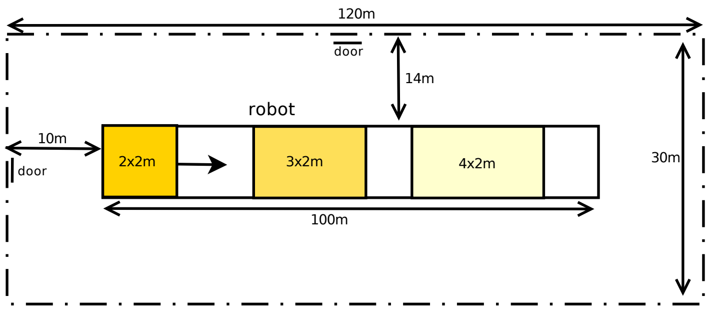


Figure E8. Possible space occupation induced by unknown speed

is the module name. A PTA transition in PRISM is started with a pair of brackets (*[]*) and optional synchronization commands in the brackets, e.g., a semaphore *i* initializes distributed building blocks simultaneously during the system initialization, and semaphores *r,y* are abbreviated from communication parameter *setRed, setYellow* to synchronize module *Control Unit* (robot controller) and module *Robot Operation* (robot actuator). ESM transitions which are labeled as real-time probabilistic actions are exported and extended with probabilistic description. Line *14* to *17* gives an example command in PRISM showing the probabilities. It declares that when *s_c2* variable equals to 2 it has *80%* possibility of going to state 5, *98%-80%* possibility of going to state 6. When state 5 is reached, guard conditions demand that clock *c_c2* must be no greater than *150* (representing the system delay in 100 microseconds) and not smaller than *160* (see also Table F1).

4 Probabilistic Spatial Property Verification

Probabilities in system models can affect the spatial behavior of systems. Depending on the specification — as provided by Reactive Blocks — we can determine areas in time and space which a system component is likely to occupy or interact with.

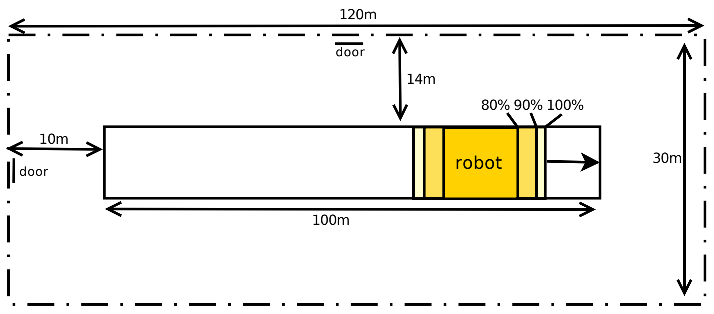


Figure E9. Space occupation and probability

In our robot system the distribution of latencies for reacting to the detection of a human can result in different areas indicating the possible positions of a robot for given time points, each one associated with a probability. Another example is varying speed. If the speed sensors of the robot are not accurate, they may come with a distribution of a possible error. Therefore the robot may accelerate to a speed slightly higher or lower than the specified 10 m/s. As depicted in Fig. E8, this inaccuracy leads to a wider area, the robot may be in at a certain point of time, and the sizes of the areas increase over the distance the robot moves. Furthermore, one can relate the area sizes also with probabilities. Following the discrete probability distribution in Table F1, in Fig. E9 we show the varying areas covered by the robot. With a probability of $80\ \%$ it will be within the orange rectangle at a selected point of time, with $90\ \%$ in the dark yellow one and with certainty in the light yellow one.

For collision analysis we may neglect probabilities that are below a certain threshold defining residual risks that one is willing to bear. In our example, we can prove that there is indeed a situation that a person running into the factory hall with a speed of $10\frac{m}{s}$, may hit the robot before it completely stopped. According to the distribution in Table F1, the risk for this, however, is not higher than $5 \cdot 10^{-14}$. Since we found out by simulating the situation that the speed of the robot at such an impact is $0.625\frac{m}{s}$ at most, the collision risk is extremely low and the impact essentially not different from the human running into a stationary object.

We implemented BeSpaceD, a tool [4] for checking spatial behavior of spatiotemporal systems as well as an input language for such systems. The implementation is done in Scala and comprises abstract datatypes that indicate spatial availability, interaction or occupation in areas in a coordinate system for time intervals or timepoints. It is possible to give parameterized specifications describing nondeterministic systems and their spatial behavior.

For this work, however, we restrict ourselves to the checking of scenarios generated from simulation runs of Reactive Blocks models. Particularly, in a simulator run we stored every five milliseconds a tuple consisting of the current time stamp and the positions of human and robot in a format readable by BeSpaceD. Thereafter, we use BeSpaceD to detect collisions and other spatial interactions for the various scenarios and probabilities. In this way, we found out the situation mentioned above that a human indeed may collide with the robot before it has stopped. We are able to learn such space-related safety issues already while modeling the system. This makes it much easier to adapt the system functionality or to impose stricter real-time properties if non-bearable situations were detected.

The specification of each spatial entity in a scenario has the form:

$time = t \longrightarrow$
  occupied spatial area with probability $p \wedge ... \wedge$ occupied spatial area with probability p'
$time = t + 1 \longrightarrow$
  occupied spatial area with probability $p \wedge ... \wedge$ occupied spatial area with probability p'

....

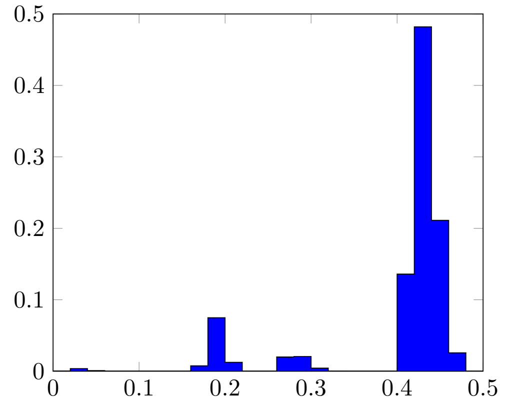


Figure E10. Probability density function for the system

$time = t + n \longrightarrow$
occupied spatial area with probability $p \wedge ... \wedge$ occupied spatial area with probability p'

Probabilities are treated as attributes to an occupied spatial area. Different means to check spatial properties — here collisions — formulated over these inputs are provided in BeSpaceD and are based on SAT and SMT solving and direct Scala implementations.

5 Probability Distributions as Verification Results

We verify probabilistic properties based on the probabilistic timed temporal logic PTCTL [25]. The probability operator $P_{=?}$ allows reasoning about numerical probabilistic values, and is supported by the so called stochastic games engine [21] in the verification tool PRISM. The results of our property verification are probability distributions. Figure E10 shows the non-accumulative probability distribution of the overall system reaction time displayed in a histogram. The *x* axis represents time values with a frequency of 0.02 s. The *y* axis shows the numerical probability value.

In our analysis, we verified a set of probabilities expressed in PTCTL as follows:

$$P_{=?}[F_{\leq T}\, ''target''](T \in [0.0, ...0.5])$$

In the above temporal specification formula, the operator F is a path operator that equals to *eventually* in LTL and can be used inside the P operator. The pattern $F_{\leq T}$ stands for "*within T time unit*". The logical expression *"target"* inside the PTA models that the parallel composed real-time probabilistic actions are indeed executed. The formula expresses the possibility that within T time units, the labeled actions are achieved.

Important for us are questions like: Is the robot reaction time no more than a designed safe time limit of 0.46 s? Checking that a reaction time of 0.46 s is enough to avoid a collision is discovered by simulation using Reactive Blocks and BeSpaceD. With PTCTL we check this property using the following formula:

$$P_{=?}[F_{\leq 4600}\, ''target'']$$

The result indicates that when a human enters the monitored area, the robot reacts within $0.46\ s$ with the probability of 99.99874114988752 %. Due to the potential severity of a collision, this number does not seem sufficient. We discussed in Sect. 4 that the impact will not be serious with a reaction time of $0.5\ s$. Of course, if the robot controller reacts within $0.47\ s$, the maximum speed at impact will be even lower (i.e., $0.125\frac{m}{s}$ at most according to our simulations) and the risk of injuries is seen to be remote. Considering this fact, we assume that the layout of our example system as sufficiently safe.

6 Related Work

Here, we present related work regarding formal methods, safety analysis, probabilities and spatial verification.

In [14] an algorithm is presented to check whether a given Markov Chain satisfies formulas of Probabilistic Computation Tree Logic (PCTL). PCTL and related verification techniques are typically applied to analyze the reliability of timed systems. The work presented in [20] presents an algorithm for the verification of probabilistic real-time systems annotated with discrete probability distributions. This can be seen as a starting point for our work.

Safety analysis is a critical phase in the development of the systems we are aiming at. In [24], a solution for incorporating safety requirements in software architecture is provided. Means facilitating Model-Driven Architecture based development and safety analysis are presented. The work presented in [11] aims at providing a modeling framework for the hazard analysis of component-based systems. The authors intended to include traditional hazard analysis techniques, e.g., Fault Tree Analysis (FTA), but found its inappropriateness for component based systems. Thus, new techniques like State Event Fault Tree (SEFT) are proposed. The new proposed model is suitable for describing stochastic behaviors with the help of existing tools such as Matlab/Simulink. These can be extended to support such models. In [23] an extension of concurrent Kleene algebras is provided. An axiomatisation for probabilistic, concurrent and nondeterministic systems is presented, and the simulation and verification of probabilistic automata can be carried out. In the past we have studied the application of probabilistic models in a theorem prover for guaranteeing fault-tolerance of embedded systems [3].

A process algebra-like formalism for describing and reasoning about spatial behavior has been introduced in [5, 6]. Process algebras come with a precise formal semantics and target the specification of highly parallel systems. Another logic-based approach to describe spatial areas is the Region Connection Calculus (RCC) [3]. It includes spatial predicates to describe the separation and connection of areas. The area of hybrid systems has features the development of different tools for reasoning and verification. SpaceEx [10] allows the modeling of continuos hybrid systems based on hybrid automata. It can be used for computing overapproximations of the space occupied by an object moving in time and space. Additionally, it is possible to model spatial behavior in more general purpose oriented verification tools in Hybrid systems (e.g., [25]).

7 Conclusion

In this paper, we proposed an approach for the model-driven development of probabilistic real-time systems with highly reusable compositional building blocks that incorporate discrete probability distributions for describing stochastic time behaviors. These extensions are used to predict system performance as well as probabilistic safety properties. In addition, we established a development tool set both for temporal and spatial probabilistic behaviors of systems. The Probabilistic Real-Time External State Machines (PRTESM) of building blocks give a straightforward view of stochastic real-time behaviors of component-based systems. Software architects and safety engineers can use this for verification and analysis. The limitations of the approach is that some intral or inter components' communications are, instead, described as messages passed and received in UML, such that such behaviors can not be captured by ESMs. Also, whether this approach is applicable depends on the abstraction level. In the future, we want to further study and emphasize the compositionality of probabilistic spatial behavior definitions, i.e., of systems composed of appropriately logically detailed subsystems. We plan to advance the description logic as well as the introduction of a probabilistic spatial behavioral type infrastructure that extends previous work [4, 5].

Acknowledgments We like to express our thanks to Song Zheng Song from the National University of Singapore (NUS) for the useful discussion during the research work.

Bibliography

[1] R. Alur, D. Dill. Automata for Modeling Real-Time Systems, in Automata, Languages and Programming, pages 322–335, vol. 443 of LNCS, Springer-Verlag, 1990, doi:10.1007/BFb0032042.

[2] B. Bennett, A. G. Cohn, F. Wolter, M. Zakharyaschev. Multi-Dimensional Modal Logic as a Framework for Spatio-Temporal Reasoning. Applied Intelligence, 17(3), Kluwer Academic Publishers, November 2002, doi:10.1023/A:1020083231504.

[3] J. O. Blech. Probabilistic Compositional Reasoning for Guaranteeing Fault Tolerance Properties. 15th International Conference On Principles Of Distributed Systems, Toulouse, France, vol. 7109 of LNCS, Springer, December 2011, doi:10.1007/978-3-642-25873-2_16.

[4] J. O. Blech. Towards a Framework for Behavioral Specifications of OSGi Components. Formal Engineering approaches to Software Components and Architectures. Electronic Proceedings in Theoretical Computer Science, 2013, doi:10.4204/EPTCS.108.6.

[5] J. O. Blech, Y. Falcone, H. Rueß, B. Schätz. Behavioral Specification based Runtime Monitors for OSGi Services. Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), vol. 7609 of LNCS, Springer-Verlag, 2012, doi:10.1007/978-3-642-34026-0_30.

[6] J. O. Blech and H. Schmidt. Towards Modeling and Checking the Spatial and Interaction Behavior of Widely Distributed Systems. Improving Systems and Software Engineering Conference, Melbourne, Sep. 2013.

[7] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). Information and Computation, 186(2), Nov. 2003, doi:10.1016/S 0890-5401(03)00137-8.

[8] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part II). Theoretical Computer Science, 322(3), pp. 517–565, Sep. 2004, doi:10.1016/j.tcs.2003.10.041.

[9] L.A. Gunawan, P. Herrmann. Compositional Verification of Application-Level Security Properties. Proceedings of the International Symposium on Engineering Secure Software and Systems (ESSoS 2013), pages 75-90, Paris, vol. 7781 of LNCS, Springer, Feb/Mar 2013, doi:doi:10.1007/978-3-642-36563-8_6.

[10] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, O. Maler. SpaceEx: Scalable Verification of Hybrid Systems. Computer Aided Verification (CAV'11), 2011, doi:doi:10.1007/978-3-642-22110-1_30.

[11] L. Grunske, B. Kaiser and Y. Papadopoulos. Model-driven safety evaluation with state-event-based component failure annotations. In CBSE, pages 33–48, vol 3489 of LNCS, Springer, 2005, doi:10.1007/11424529_3.

[12] F. Han, P. Herrmann, and H. Le. Modeling and verifying real-time properties of reactive systems.' 18th International Conference on Engineering of Complex Computer Systems (ICECCS), 2013, doi:10.1109/ICECCS.2013.13.

[13] F. Han. P. Herrmann. Modeling real-time system performance with respect to scheduling analysis. Proceedings of the 6th IEEE International Conference on Ubi-Media Computing, Nov 2013.

[14] H. Hansson and B. Jonsson. A Logic for Reasoning about Time and Reliability. Formal Aspects of Computing, 6(5). 1994, doi:10.1007/BF01211866.

[15] D. Henriques, J.G. Martins, P. Zuliani, A. Platzer, E.M. Clarke. Statistical Model Checking for Markov Decision Processes. Quantitative Evaluation of Systems (QEST), 2012 Ninth International Conference on , vol., no., pp.84,93, 17-20 Sept. 2012, doi:10.1109/QEST.2012.19.

[16] F.A. Kraemer, P. Herrmann. Automated Encapsulation of UML Activities for Incremental Development and Verification. Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MODELS 2009), pages 571-585, Denver, vol. 5795 of LNCS, Springer-Verlag, Oct. 2009, doi:10.1007/978-3-642-04425-0_44.

[17] F. A. Kraemer, V. Slåtten, and P. Herrmann. Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services. Journal of Systems and Software. vol. 82, no. 12, pp. 2068–2080, December 2009, doi:10.1016/j.jss.2009.06.057.

[18] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07), Eds. M. Bernardo and J. Hillston, pp. 220–27, vol. 4486 of LNCS (Tutorial Volume), Springer, 2007, doi:10.1007/978-3-540-72522-0_6.

[19] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. Proc. 23rd International Conference on Computer Aided Verification (CAV'11), pp. 585–59, vol. 6806 of LNCS, Springer, 2011, doi:10.1007/978-3-642-22110-1_47.

[20] M. Kwiatkowska, G. Norman, R. Segala and J. Sproston. Automatic Verification of Real-time Systems with Discrete Probability Distributions. Theoretical Computer Science, 282, pages 101-150. June 2002, doi:10.1007/3-540-48778-6_5.

[21] M. Kwiatkowska, G. Norman and D. Parker. Stochastic Games for Verification of Probabilistic Timed Automata. Technical report RR-09-05, Oxford University Computing Laboratory. June 2009, doi:10.1007/978-3-642-04368-0_17.

[22] M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant Systems (FORMATS/FTRTFT'04), Eds. Y. Lakhnech and S. Yovine, pp. 293–308, vol. 3253 of LNCS, Springer, 2004, doi:10.1016/j.ic.2007.01.004.

[23] A. McIver, T. Rabehaja, and G. Struth. Probabilistic concurrent kleene algebra. Proceedings 11th International Workshop on Quantitative Aspects of Programming Languages and Systems, Rome, 23rd-24th March 2013, Eds. L. Bortolussi and H. Wiklicky, pp. 97–115, vol. 117 of Electronic Proceedings in Theoretical Computer Science, Open Publishing Association, 2013, doi:10.4204/EPTCS.117.7.

[24] M.A.de Miguel, J. F. Briones, J. P. Silva, A, Alonso. Integration of safety analysis in model-driven software development. Software, pp.260-280, vol.2, no.3, , IET, June 2008, doi:10.1049/iet-sen:20070050.

[25] D. Parker. Verification of Probabilistic Real-time Systems. Proc. 2013 Real-time Systems Summer School (ETR'13). August 2013.

[26] A. Platzer, J-D. Quesel. KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description). International Joint Conference on Automated Reasoning, pp. 171–178, LNCS 5195, Springer, 2008, doi:10.1007/978-3-540-71070-7_15.

[27] V. Slåtten, F. A. Kraemer, and P. Herrmann. Towards Automatic Generation of Formal Specifications to Validate and Verify Reliable Distributed Systems: A Method Exemplified by an Industrial Case Study. Proceedings of the 10th ACM International Conference on Generative Programming and Component Engineering (GPCE11), pp. 147–156, ACM, 2011, doi:10.1145/2047862.2047888.

PAPER F

A Model-based Toolchain to Verify Spatial Behavior of Cyber-Physical Systems

Peter Herrmann , Jan Olaf Blech, Fenglin Han and Heinz Schmidt

In proceedings of The 2014 Asia-Pacific Services Computing Conference (APSCC), 4-6$_{th}$ Dec, 2014, Fuzhou, China.

This paper was awarded the Best-Track Paper Award of the Special Track on Reliability Technologies and Tools for Services-Based Systems.

In International Journal of Web Services Research (IJWSR), volume 13 No. 1 (2016).

PAPER G

Model-based Engineering and Analysis of Space-aware Systems Communicating via IEEE 802.11

Fenglin Han, Jan Olaf Blech, Peter Herrmann and Heinz Schmidt

In 39th Annual International Computers, Software & Applications Conference (COMPSAC 2015). pages 638-646, Taichung, Taiwan, July 2015, IEEE Computer.

A UNIFIED MODEL-BASED DEVELOPMENT FRAMEWORK FOR SPACE-AWARE SYSTEMS COMMUNICATING VIA IEEE 802.11

Fenglin Han
Norwegian University of Science and Technology, Trondheim, Norway
sih@item.ntnu.no

Jan Olaf Blech
RMIT University, Melbourne, Australia
janolaf.blech@rmit.edu.au

Peter Herrmann
Norwegian University of Science and Technology, Trondheim, Norway
herrmann@item.ntnu.no

Heinz Schmidt
RMIT University, Melbourne, Australia
heinz.schmidt@rmit.edu.au

Abstract We propose a model-driven development approach for autonomous control systems with emphasis on the physical space and the communication via wireless connections. In particular, we combine model-based engineering with simulation and emulation techniques for mobile communication. The design and implementation is done using our Reactive Blocks Framework. For the mobile communication we use the popular IEEE 802.11 WLAN protocol which is simulated using simulation tools in order to get estimations of connection delays. The spatial constraints are verified with our BeSpaceD tool. As an example, we present the design and verification of autonomous robots performing services in a large factory hall and coordinating by means of wireless communication which is based on several access points.

Keywords: Model-Driven Development, IEEE 802.11 WLAN, Spatial Verification.

1 Introduction

Recent technological development leads to a growing number of systems in which various mobile components act autonomously in a shared physical space. Examples for such systems abound in domains such as robotics, aeronautics, and automotive manufacturing. The autonomous systems must not interfere with each other in an undesired way which could result in accidents. Further, in many applications they form collaborative groups in order to perform joint tasks (e.g., several robots transport a heavy workpiece together on a construction site or coordinate their actions for warehouse automation). To achieve such a collaborative behavior, the components need to continuously interact with each other via wireless connections and the communication has to keep hard real-time limits. When developing such a system, one therefore has to consider communication-related issues (e.g., bandwidth limits occurring when many components are close together such that they are handled by a single access point). Szanto et al. show that the performance of a teleoperated robot system is strongly influenced by the quality of the communication environment [26].

Many industrial methods, e.g., Matlab/Simulink- and IEC 61131-based approaches, seek to build control systems solely from a model-based software engineering point of view. Others, e.g., OPNET [9], focus only on the communication network design and analysis.

In this paper, we propose bridging these two areas. We study and evaluate a method for real-time control applications of wireless interconnected devices that takes both wireless network response times and spatiotemporal properties into account. To this end, we combine network simulation using Jemula [4] with spatial constraint solving using BeSpaceD [6, 7]. In extension of this previous work,

1 we use constraint solutions to provide parameters for network simulations;

2 the results of the network simulations including failure probabilities and other statistical properties are then used to input further constraints into the constraint solver. For instance, if an access point fails, the neighboring ones may take over parts of its traffic, but at the expense of heavier loads.

3 The expected communication delays can be simulated with Jemula when one or more access points fail.

We also continue to use our model-based software engineering method Reactive Blocks[1] that allows us to specify system models by composing reusable sub-models, so-called building blocks [17].

This approach enables us to create controllers for mobile components using the combination of these three methods. Our contribution is the combination of these methods as well as the exchange of models and the flow of analysis results.

Moreover, we applied BeSpaceD to formally analyze whether relevant spatial safety properties (e.g., no accidents) are fulfilled if the system reacts within certain time limits [13].

The interaction of the three methods and tools is depicted in Fig. G1. In the current version of the toolchain, we restrict ourselves to the popular IEEE 802.11 series of WLAN protocols as interconnection technology and use the open source IEEE 802.11 simulator/emulator Jemula [4] to simulate WLAN usage in various stages of a scenario getting significant predictions of the communication delay. So, if we want to verify spatial properties with BeSpaceD and the proofs need knowledge about worst-case or average communication delays, we first create and execute appropriate simulation-runs. The results of the runs can thereafter be used in the BeSpaceD proofs. This combination of the tools allows us to make useful predictions about the communication infrastructure of an embedded system already on the modeling level. If spatial properties cannot be proven, it is usually much cheaper to adapt the system infrastructure and the functionality of the mobile components on the modeling level than later when the system has already been implemented.

In Sect. 2, we introduce the background of our approach. To facilitate understanding of the approach, we describe a mobile robot system-based scenario in Sect. 3. The detailed control module description is discussed in Sect. 4 followed by the definition of spatial properties in Sect. 5. The WLAN communication simulation results for the scenario are introduced in Sect. 6. In Sect. 7, we explain how the simulation can be combined with the spatial analysis. The text is completed by a discussion about related work and a conclusion.

2 Background

The technology used in this work consists of the three tools Reactive Blocks, BeSpaceD and Jemula mentioned above. Further, relevant characteristics of the IEEE 802.11 protocol are discussed.

[1]Until recently Reactive Blocks was named Arctis.

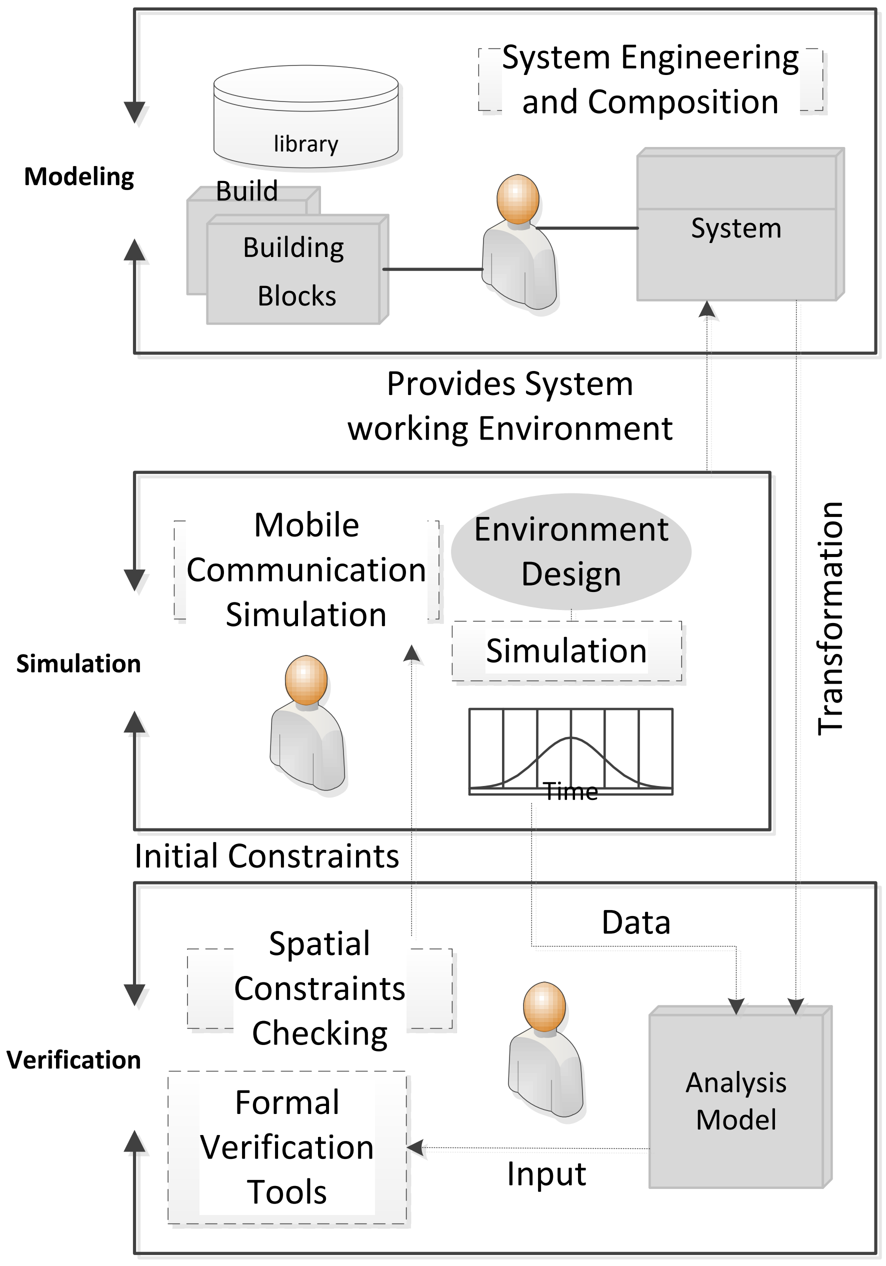

System Engineering and Composition
library
Modeling
Build
Building Blocks
System
Provides System working Environment
Mobile Communication Simulation
Environment Design
Simulation
Simulation
Time
Transformation
Initial Constraints
Data
Spatial Constraints Checking
Verification
Analysis Model
Formal Verification Tools
Input

Figure G1. Summary of the Approach.

2.1 Reactive Blocks

The model-driven engineering method Reactive Blocks is a tool-supported approach for developing reactive concurrent software systems [22]. The systems are composed from models of reusable software components, so-called *building blocks*. The main ingredients of a building block are a UML 2.x activity diagram and an abstract *External State Machine* (ESM) [19]. Similar to a Petri net, the activity diagram models the detailed implementation logic as a token flow. To allow formal analysis, we supplemented the activity diagrams with a formal semantics allowing to order the token flows into reactive run-to-completion steps (see [20]). The ESM shows the abbreviated interface behavior of the building block as an abstract UML 2.x state machine. The concept enables to specify recurrent sub-functionality by separate building blocks that can be specified once, stored in model libraries and reused in a drag-and-drop fashion in models. System models can be automatically analyzed for functional errors by a built-in model checker [22]. Further, executable Java code can be automatically generated [21]. The tool-set is now marketed by *BitReactive AS*[2] as Reactive Blocks and networked embedded devices is a prominent application domain for it.

An extension of the tool supports also the analysis of probabilistic real-time performance and safety issues. In particular, the time and probabilism properties can be formalized and analyzed using probabilistic timed automata-based formal verification [11, 13]. The formal analysis and verification is based on the two verification tools UPPAAL [3] and PRISM [23]. We established two extensions of the ESMs called *Real-Time External State Machine* (RTESM) [11, 12] and *Probabilistic Real-Time External State Machine* (PRTE
SM) [13] to model time-constrained behavior. This allows us to check hard real-time properties, e.g., proving that a system leaves a certain state within a period of time in order to carry out some safety preserving actions.

2.2 BeSpaceD

In [6, 7], we introduce BeSpaceD as a tool framework for specifying behavior of distributed systems and formally reasoning about them. BeSpaceD emphasizes on spatial behavior but is not restricted to this. It allows the verification of safety properties such as the absence of physical collisions between interacting robots and obstacles, the coverage of sensor ranges, or WLAN ranges. Specification is done using abstract datatypes out of a development environment supporting the Scala programming language. The abstract datatypes can be generated by Scala programs or by instantiation of other software. Checking and reasoning in BeSpaceD is realized using library functions creating verification goals. Verification goals are solved by standard tools such as SAT and SMT solvers or by specialized algorithms. For the purpose of this paper, specifications of spatial problems are done using the BeSpaceD constructs provided in a library inside code written in the Scala programming language.

2.3 IEEE 802.11 WLAN Delay Analysis

The IEEE 802.11 wireless local area network (WLAN) protocol is widely used for enterprise, home and public access networks. It consists of several protocol variations that define specifications for Media Access Control (MAC) and Physical (PHY) layer specifications. In the MAC layer, there are mainly two coordination functions, the *Distributed Coordination Function* (DCF) and the *Point Coordination Function* (PCF) that specify the media access and collision control. DCF is a distributed medium access scheme based on Carrier Sense Multiple Access using a Collision Avoidance (CSMA/CA) scheme with binary slotted exponential backoff. In the basic IEEE 802.11 MAC protocol using DCF, a station determines individually when to access the media. The station service responsible for information exchange is referred to as MAC Service Data Unit delivery (MSDU delivery). In contrast, PCF is a centralized media access function, in which the access point grants access to the different stations by polling.

Due to the competitive media access, DCF has the greatest impact on the performance of IEEE 802.11 wireless protocols. In our simulation we use the so called *Request/Clear To Send* (RTS/CTS) access mode which is the general mechanism to reserve the wireless communication channel in DCF. The WLAN latency is mainly composed of two parts, the transmission delay and the backoff delay. The transmission delay refers to the latency of transferring packets over the net and depends on the packet

[2]http://www.bitreactive.com.

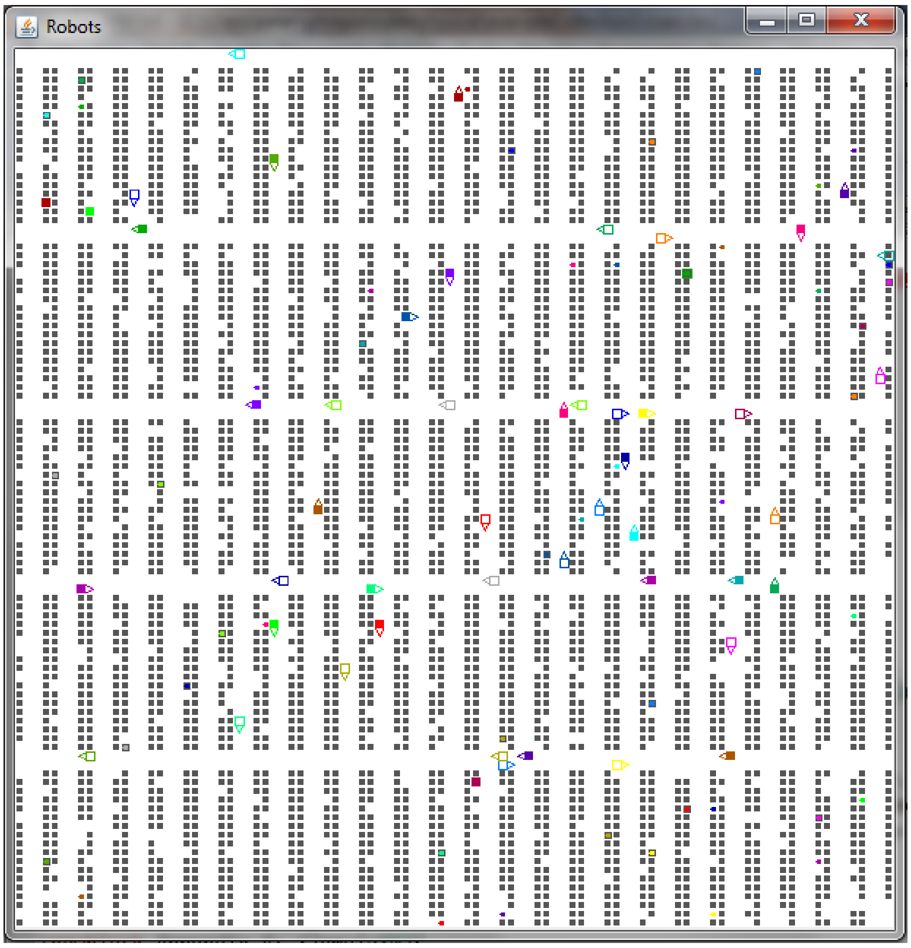


Figure G2. Animation of the Warehouse Robot System with 50 Robots.

size (number of bits) and the transmission rate. The backoff delay refers to the lag of time a station needs to access the WLAN. The IEEE 802.11 MAC DCF uses the so-called binary slotted exponential backoff [5] to estimate the backoff delay. That is an analytical model computing the saturation throughput performance of DCF as a *stochastic* process. The model is easy to understand but can be hardly analyzed by machines due to the state space explosion problem. Therefore, we use emulators to simulate the communication collision and delay of stations to discover how the number of mobile stations can affect the delays and, in consequence, the overall network communication.

We use Jemula802 and its kernel Jemula [4], which are open-source Java-based emulation tools to model IEEE 802-based communication. The Jemula emulation software has an event-based architecture and maintains an XML interface for the configuration of networks and systems. Like Reactive Blocks and BeSpaceD, Jemula is developed under the Eclipse framework.

3 Motivating Scenario

In this section, we introduce a motivating scenario comprising a mobile distributed robot system followed by a closer discussion of its communication infrastructure.

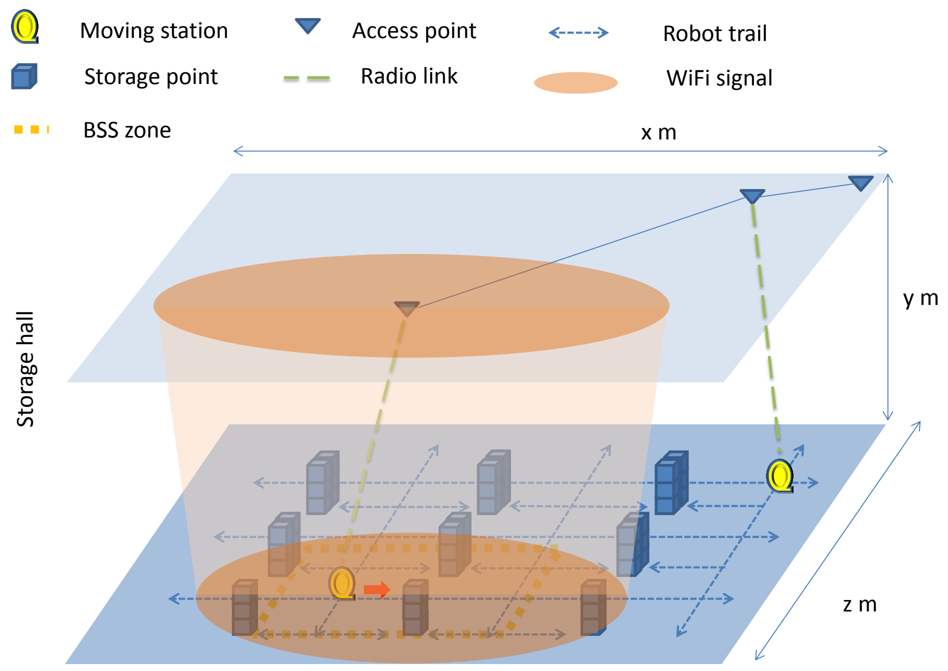


Figure G3. Communication Infrastructure in the Storage Hall.

3.1 A Warehouse Robot System

We created a simulator of a mobile robot fulfilment system that contains multiple robots and handles goods in a warehouse. Our scenario is inspired by the Kiva robot system[3]. As depicted in Fig. G2, the robots fetch pallets (black rectangles) and transport them to other designated positions in the warehouse. Robots currently transporting a pallet are shown in the animation as filled colored rectangles while the unfilled ones refer to robots not carrying a pallet. The robots run on a line grid marking the floor. Lines are spaced $1\ m$ apart. Whenever a robot leaves a crossing, it signals its path to the next crossing to all the other robots in order to prevent that another robot heads towards the same crossing. When a new crossing is reached, another message to the other robots indicates that the crossing from which the robot left, is freed.

Due to the communication delay, nevertheless, up to four robots may be on their way to the same crossing. As soon as a robot learns about such a conflict by receiving the respective signal from one of the other robots, it immediately stops and moves back to the crossing from which it was coming. Due to the physical layout of the system, the combined communication delay and reaction time leading to an emergency stop needs to take place within a second to avoid collisions. We showed in [11, 12] how maximum reaction times can be computed based on the system model. For our scenario, we found out that a reaction time of 200 ms is sufficient to stop the robot such that the maximum communication delay is 800 ms. The proof under which circumstances this maximum delay can be guaranteed by the system, is discussed later in this paper.

[3]https://www.youtube.com/watch?v=lWsMdN7HMuA.

3.2 WLAN topology and environment

We use a WLAN for the coordination and scheduling of the moving robots. We sketch this in Fig. G3. The working space of the robots is modeled as a plane of width x, height y, and depth z. We assume that several access points are installed in the ceiling. The scenario is simplified by using the following assumptions:

1 We assume that neither the pallets and their content infer with the radio signals nor that there are obstacles between the access points and the robots. Thus, we suppose idealized conditions for the quality of the radio signals.

2 The access point antennas in the warehouse are arranged in a way that every robot is always in the range of one of them. To achieve that, we partition the warehouse into access areas and each access area is completely covered by the circular range of an access point. This will be discussed in detail in Sect. 5.

3 Since robots can only physically interfere with each other if they are in close proximity, they will be either covered by the same access point or are at the border between two access points. Thus, a robot only needs to consider other robots and objects covered by the same or by neighboring access points. We reflect that in our simulations introduced in Sect. 6.

4 Modeling the Controller in Reactive Blocks

Since the robot scenario consists of several robots of the same sort, we define the robot controller as a multi-session building block [18] in Reactive Blocks. This allows us to model an arbitrary number of control entities that can interact with each other as well as with their common environment. Figure G4 depicts the UML activity of the multiple-session building block that we named *Robot* at the top while its ESM is found in at the bottom. Each robot has a unique identifier stored in the variable *me*. The other variable *data* stores information such as the current position and direction of the robot, the pathes of other robots, and the positions of the pallets in the warehouse.

Robot contains three internal building blocks that each have their own UML activities and ESMs. *TimeStampOccupyBoxManager* is taken from a library and manages the handling of data types carrying information about spatial properties and time of a robot. The building block *HybridKIVASystem* contains the basic logic of the robot control. Whenever the robot either reaches its destination or a crossing from which it can continue in various directions, a token is triggered passing pin *callDecision* of block *HybridKIVASystem*. This token is duplicated at the fork behind this pin and one copy forwards to operation *newStop* in which a Java method is executed that creates a message to inform the other robots that the crossing, from which the current one is reached, has been vacated. The token is forwarded to the parameter node *sendNewNext* enabling the environment of building block *Robot* to trigger a WLAN message to all other robots. The other token copy is forwarded to the building block *NextPosition* which contains the routing algorithm of the robot.

If the robot did not yet reach its destination, in *NextPosition* the new path is computed and forwarded via pin *nextData* to block *HybridKIVASystem*. In parallel, a flow leaves pin *sendNewNext* and forwards to a set-method for the local variable *data* that contains relevant data of the robot and its environment. Thereafter, a data unit is generated to inform the other robots of the next crossing to which the robot proceeds. The data unit is forwarded to parameter node *sendNewNext* in order to instantiate the according WLAN communication. If the robot reached its destination, a flow is issued passing via parameter node *callNewDest* to the system control which might assign a new task to the robot. The answer from the system control is received via a flow through *getNewDest*. As indicated by the filter *to all*, the new destination is sent to all robots since each one stores which pallets are currently transported by a robot.

Messages indicating the current positions and paths of other robots are received via parameter node *rcvNewNext* from where they proceed to operation *othersNewOccupation*. In the corresponding Java method, the received data is stored in variable *data*. The message is forwarded to building block *NextPosition* to be checked if the other robot is on a collision course with the local one. In this case, a token is

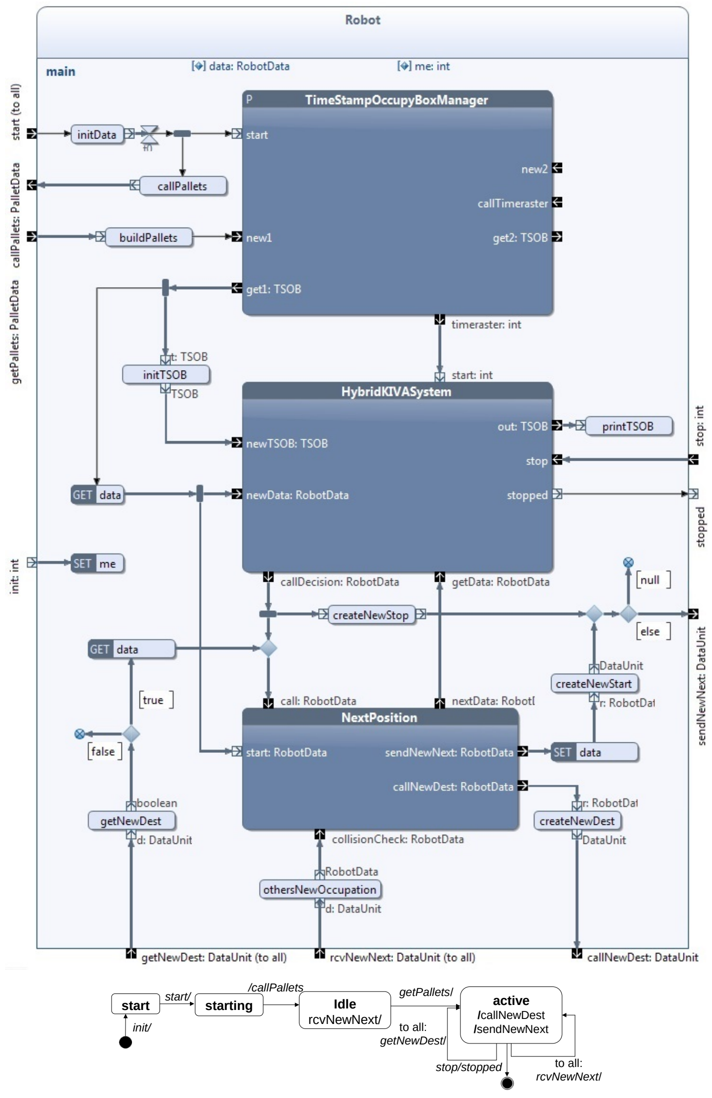


Figure G4. Multi-session Building Block *Robot*.

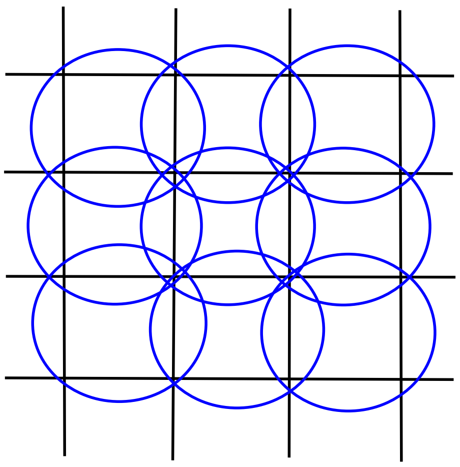


Figure G5. Spatial Arrangement of Access Points.

sent via pin *nextData* to block *HybridKIVASystem* that causes the emergency stop of the robot as well as the retreat to the crossing from where the robot left.

The *Robot* building block is the core communication function and decision making component for the movement control of the transportation robot. Due to the saturation of network traffic, this module is under heavy burden. In order to define the burden of the module, especially the saturation of state *active* in the ESM of the *Robot* block (see Fig. G4), we use a Markov chain-based performance model of the control system in our simulation.

5 Communication Access Ranges

The warehouse layout discussed in Sect. 3.2 raises a number of spatial issues that have to be addressed by the overall system design:

1 The access points have a limited range and we have to ensure that every point in the storage hall in which robots may act, is sufficiently covered by access points. That holds particularly, when unlike to our assumptions in Sect. 3.2, the storage hall contains obstacles which may distort radio signals.

2 An access point may fail with a certain probability. In this case, some robots in the coverage area of the broken access point may be covered by the neighboring access points while other robots may experience communication loss. In the former case, the access points taking over may get saturated. We verify properties of saturation, spatial aspects and failure probability to foster risk analysis. To guarantee high availability of continuous communication access for all robots, we require this failure probability to be below a minimal threshold.

3 Robots are moving through the hall and it may be that several of them are in proximity to each other. That does not only increase the probability of accidents between robots but might also increase the communication delay of the access points in this zone. We have to find out if there is sufficient bandwidth to guarantee certain maximum communication delays even if many robots are close together.

BeSpaceD [6, 7] allows us to establish spatial models for access point ranges including probabilities for, e.g., failures. For example, we can define a higher likelihood of a communication failure if a robot is farer away of the nearest access point. We assume that the robots keep a perfect, error-free connection with an access point if they are within a certain limited radius but that the QoS deteriorates when the robot is outside this radius. Furthermore, we can formalize behavioral properties of robots like positions and paths and also situation-dependent constraints on the channels [13]. Ranges of access points tend to have a circular shape. But we can also partition the ranges in other shapes like the rectangular partitioning that is shown in Figure G5. Here, a square describes an area fully covered by one access point. Particularly the edges of a rectangle may also be covered by other access points.

For our robot scenario, we used the BeSpaceD tool to check the above mentioned consistency and safety properties regarding the coverage. In particular, we checked consistency properties, e.g., is there a sufficient coverage for a given number of robots and safety properties: Given a probability distribution for communication delays, can we guarantee an upper bound for communication time between an access point and a robot? This is closer discussed in Sect. 7.

Below, we list a small code fragment written in Scala for the specification of a simplified coverage problem consisting of circular access point ranges for a rectangular factory hall at a given point of time below:

```
def coverage =
  IMPLIES (AND(Owner ("WLAN_RANGE"),
  Prob(1.0)), BIGAND(
             OccupyCircle(15,15,40)::
             OccupyCircle(35,15,43)::
             ...
             OccupyCircle(65,130,42)::
             Nil ));

  def factoryhallarea =
   IMPLIES (AND(Owner ("FactoryHall"),
   Event("Operation")),
      OccupyBox(10,10,100,150)
  );
```

Space occupation is provided for a particular aspect (e.g., the range of an access point `WLAN_RANGE`) and the probability that communication is successful. Moreover, for mobile components the space occupation varies depending on the time. Thus, by using spatiotemporal models for both the access point ranges and for robots, one can for instance prove with BeSpaceD if a robot is always covered by circles that have probability 1.0.

6 Simulation Results

The second and third spatial properties mentioned above can only be proven if the WLAN can guarantee a certain maximum communication delay. We use Jemula802 [4] to simulate the IEEE 802.11a physical layer (PHY) at the wave band of 5 GHz that allows up to a transmission rate of 54 Mb/s. Each mobile station, i.e., each robot, uses an omnidirectional antenna to send and receive the control and data packets. We assume that the maximum packet size of the control data is 200 bytes which corresponds to the fact that the identifier of the robot as well as information about the reached and vacated crossings are transported. Jemula802 emits packets at time intervals following a negative exponential distribution. For our example, the tool generates a mean load of 0.9 Mb/s. The packet size varies by a small amount which is uniformly distributed.

We configured a traffic generator for each mobile robot under the same access point. The generator emits network packets in the form of MAC Service Data Unit (MSDU) deliveries [25] and the simulated transmission time corresponds to the network communication latency for the robots. We plotted the communication delays simulated for 40 s and simulated several scenarios with different numbers of robots. Figure G6 shows the result of this simulation when an access point interconnects 50 mobile robots. Figure G7 depicts the delay distribution histogram of the simulation results. The simulation generates 400 MSDU packets, and we used the *matlab* tool to get several useful indices: The maximum delay is 535.7 ms and the mean or expected value of the communication delay is 142.9 ms. Compared with the maximum delay of 281.3 ms, and the expected value 76.7 ms when we consider only 20 robots communicating via the same access point, the delay shows a significant growth. Thus, while the maximum communication delay is significant, it is sufficiently below the maximum acceptable delay time of 800 ms necessary to avoid an accident between robots. The same result was also found in the other simulations

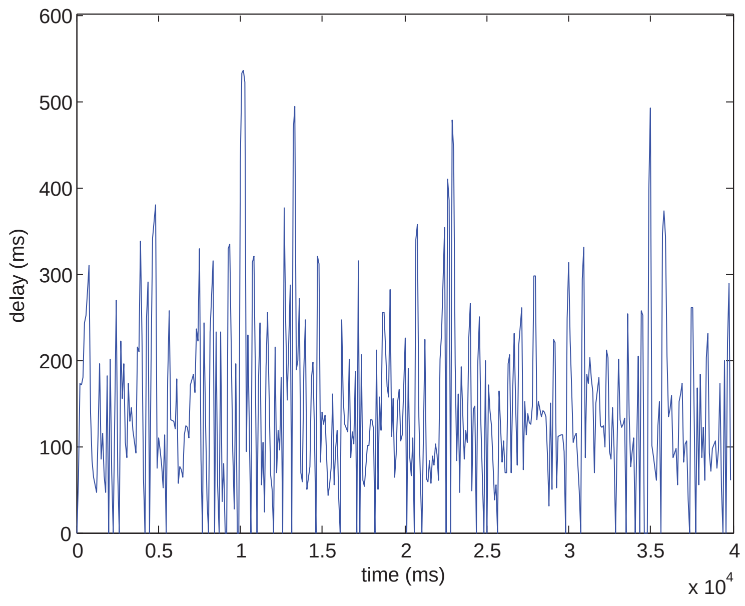


Figure G6. MSDU Delivery Delay Simulation.

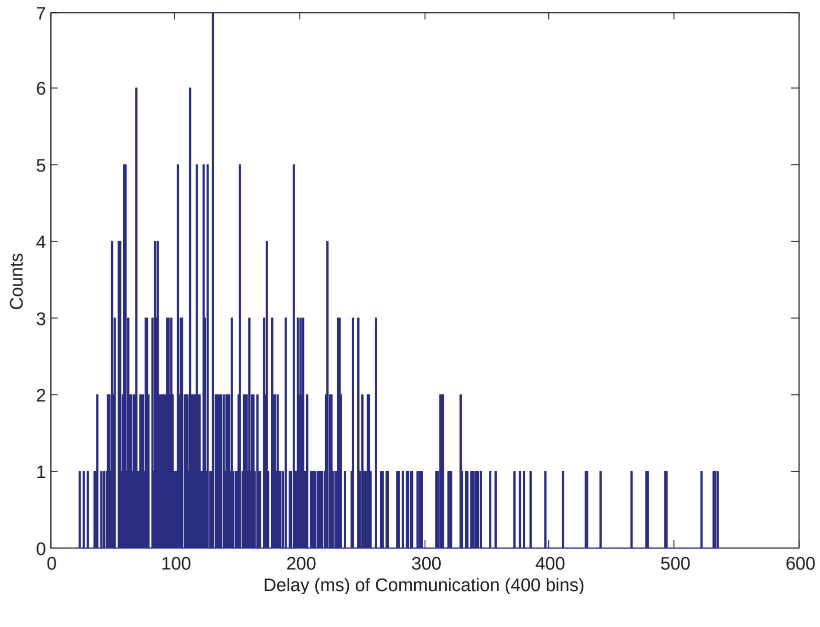


Figure G7. Communication Delay Distribution.

with different parameters, e.g., numbers of robots or size of access point ranges. Since the throughput of a wireless channel is affected by multiple factors, e.g, size of delivered MSDUs, PHY layer modes used, and bandwidth offered to the communication channel, the mobile stations are adjustable according to need.

7 Combining the Results

Our underlying method combines spatial constraint specification and solving with network topology and packet traffic simulation. On the one hand, we use spatial analysis results to parameterize the network simulation. On the other hand, we take simulation results including failure probabilities into a further spatial analysis.

BeSpaceD analyzes the satisfaction of spatial constraints listed in Sec. 5, such as sufficient coverage of the warehouse by access points so that all robots are always in range. These analysis results are used to prime the Jemula simulations. For example, we determine the topology of base stations including the number of channels available in a certain area. This, in turn, determines the parameters for and the number of Jemula runs necessary to cover all relevant use cases.

For further constraint analysis, BeSpaceD then takes the Jemula simulation results as input. For instance, if an access point fails, the neighboring ones may take over parts of its traffic, but at the expense of heavier loads. The expected communication delays can be simulated with Jemula when one or more access points fail. In BeSpaceD, we can now use these simulation results together with the probabilities that we assume for the failures of access points as well as spatial properties (e.g., the likelihood that a robot in the zone of a failed access point is at a place also covered by one of its neighbors) to verify if the overall probability that all robots can communicate with each other within a certain period of time, is greater or equal a certain value p. The value p (e.g., 99.999% that a data exchange is completed within 800 ms) helps to estimate the average risk of costs caused by malfunctions resulting in accidents. BeSpaceD was used to check several scenarios based on Jemula input.

Based on the risk analysis, one can decide if other protective mechanisms are necessary, for instance a function in the robot control logic that continuously checks if an active connection is available and immediately stops the robot if that is not the case.

For the third property of Sect. 5, i.e., the check if there is sufficient bandwidth when several robots are in a certain area, the Jemula simulations are also used. Particularly, we simulate up to which number of robots the maximum communication delay is still bearable.

8 Related Work

The surveys in [16, 24] show that due to the memory and time cost, simulation-based verification is popular in industry and software practice. It is often seen as simple and straightforward. Dill, however, states that simulation-based verification does not cope with increasing system complexity since it is getting more and more difficult to select suitable test cases [10]. Since formal verification is sometimes too complex to be automatically executed by machines and highly laborious when carried out manually, we seek an approach combining simulation and formal verification to achieve high quality software verification results in an acceptable period of time.

For the translation from state machine-based component models to Petri net analysis, Gomes et al. proposed a set of translation strategies from state machines using the history attribute to a class of non-autonomous Petri nets named Input-Output Place Transition Nets (IOPT nets) [2].

There are plenty of network simulation tools available and used widely in research. Besides Jemula, simulation and emulation tools for IEEE 802.11 wireless networks also comprise ns2 [15], OPNET [9], and NCTUns [27]. Experimental research using these tools shows that the performance of a teleoperated robot system is strongly influenced by the quality of the communication environment [26].

The IEEE 802.11 series of wireless protocols has several flavors in application to industrial robotics, among which IEEE 802.11a is considered as the most suitable existing solution. An extensive survey on wireless sensor network emulators and simulators is discussed in [14]. In [1], an experimental assessment of indoor propagation of WiFi signals with access points as transmitters operating at a frequency of 2.4 *GHz* is undertaken. The power density distribution of the access point antenna is achieved by movable laptop computers comprising wifihopper software. The measured and simulated results are compared to get a correlation coefficient factor (ρ), which indicates a good agreement between measurement and simulation results.

9 Conclusion and Future Work

In this paper, we studied and evaluated a method for real-time control applications of wireless interconnected devices. Our method takes wireless network response times and spatiotemporal properties into account. We examplified this by using mobile robots carrying out tasks in a warehouse. The ingredients of our method comprise the modeling of the system software and the simulation and analysis of the local access point networks integrated with spatial constraint solving.

In the next step, we seek further integration of the system development with the simulation and verification tools. In particular, we are aiming at directly encapsulating the simulation generator into a separate building block in our Reactive Blocks environment.

Acknowledgment We express our gratitude to Professor Yuming Jiang for his useful suggestions and discussions during the research work.

Bibliography

[1] B. M. Abaoy and K. M. Quboa. Environmental Safety Standards and WLAN Indoor Propagation. *20th Telecommunications Forum*, pp. 13–16, 2012.

[2] Pais, R. and Gomes, L. and Barros, J.-P. From UML state machines to Petri nets: History attribute translation strategies In IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society, pp. 3776-3781, Nov 2011.

[3] J. Bengtsson, F. Larsson, P. Pettersson, W. Yi, P. Christensen, J. Jensen, P. Jensen, K. Larsen, and T. Sorensen. UPPAAL: A Tool Suite for Validation and Verification of Real-Time Systems, In *Hybrid Systems III*, LNCS 1066, pp. 232–243, Springer-Verlag, 1996.

[4] L. Berlemann and S. Mangold. Appendix A: Jemula802. *Cognitive Radio and Dynamic Spectrum Access*, John Wiley & Sons, 2009.

[5] G. Bianchi. Performance Analysis of the IEEE 802.11 Distributed Coordination Function. *IEEE Journal on Selected Areas in Communication*, 18(3):535–547, 2006.

[6] J. O. Blech and H. Schmidt. BeSpaceD: Towards a Tool Framework and Methodology for the Specification and Verification of Spatial Behavior of Distributed Software Component Systems. *arXiv.org*, http://arxiv.org/abs/1404.3537, 2014.

[7] J. O. Blech and H. Schmidt. Towards Modeling and Checking the Spatial and Interaction Behavior of Widely Distributed Systems. *Improving Systems and Software Engineering Conference*, 2013.

[8] R. Calcagno, F. Rusina, F. Deregibus, A. S. Vincentelli, and A. Bonivento. Application of Wireless Technologies in Automotive Production Systems. *VDI Berichte*, 1956:57–58, 2006.

[9] X. Chang. Network simulations with OPNET. In *Proc. 31st Conference on Winter Simulation: Simulation — a Bridge to the Future (WSC'99)*, vol. 1, ACM, pp. 307–314.

[10] D. L. Dill. What's between Simulation and Formal Verification? In *Proc. of Design Automation Conference*, pp. 328–329, 1998.

[11] F. Han, P. Herrmann and H. Le. Modeling and Verifying Real-Time Properties of Reactive Systems. *Engineering of Complex Computer Systems (ICECCS)*, pp. 14–23, IEEE, 2013.

[12] F. Han and P. Herrmann. Modeling Real-Time System Performance with Respect to Scheduling Analysis. In *6th IEEE International Conference on Ubi-Media Computing*, pp. 663–671, IEEE, 2013.

[13] F. Han, J. O. Blech, P. Herrmann, and H. Schmidt. Towards Verifying Safety Properties of Real-Time Probability Systems. *Formal Engineering approaches to Software Components and Architectures (FESCA)*, EPTCS, 2014.

[14] M. Imran, A. M. Said and H. Hasbullah. A Survey of Simulators, Emulators and Testbeds for Wireless Sensor Networks. *International Symposium in Information Technology* , vol. 2, pp. 897–902, 2010.

[15] T. Issaryakul and E. Hossain. *Introduction to Network Simulator NS2*. 2nd Edition, Springer-Verlag, 2012.

[16] J. Jose and S. A. Basheer A Comparison of Assertion Based Formal Verification with Coverage driven Constrained Random Simulation, Experience on a Legacy IP. Wipro Technologies Reports.

[17] F. A. Kraemer. *Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks*. PhD thesis, Norwegian University of Science and Technology, 2008.

[18] F. A. Kraemer, R. Bræk and P. Herrmann. Synthesizing Components with Sessions from Collaboration-Oriented Service Specifications. *SDL-Forum*, LNCS 4745, pp. 166–185. Springer-Verlag, 2007.

[19] F. A. Kraemer and P. Herrmann. Automated Encapsulation of UML Activities for Incremental Development and Verification. In *Model Driven Engineering Languages and Systems (MoDELS)*, LNCS 5795, pp. 571–585. Springer-Verlag, 2009.

[20] F. A. Kraemer and P. Herrmann. Reactive Semantics for Distributed UML Activities. *Joint WG6.1 International Conference (FMOODS) and WG6.1 International Conference (FORTE)*, LNCS 6117, pp. 17–31, Springer-Verlag, 2010.

[21] F. A. Kraemer, P. Herrmann, R. Bræk. Aligning UML 2.0 State Machines and Temporal Logic for the Efficient Execution of Services. *Distributed Objects and Applications (DOA06)*, LNCS 4276, pp. 1613–1632, Springer-Verlag, 2006.

[22] F. A. Kraemer, V. Slåtten and P. Herrmann. Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services. *Journal of Systems and Software*, 82(12):2068–2080, 2009.

[23] M. Kwiatkowska, G. Norman and D. Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. *Computer Aided Verification (CAV'11)*, LNCS 6806, pp. 585–59, Springer-Verlag, 2011.

[24] William K. Lam *Hardware Design Verification: Simulation and Formal Method-Based Approaches*. Prentice Hall, 2005.

[25] S. Mangold, S. Choi, G.R. Hiertz, O. Klein, and B. Walke. Analysis of IEEE 802.11e for QoS Support in Wireless LANs. *Wireless Communications, IEEE*, 10(6):40–50, 2003.

[26] Z. Szanto, L. Marton, P. Haller, and S. Gyorgy. Performance Analysis of WLAN based Mobile Robot Teleoperation. *Intelligent Computer Communication and Processing (ICCP)*, pp. 299–305, IEEE, 2013.

[27] S.-Y. Wang and C.-C. Lin. NCTUns 5.0: A Network Simulator for IEEE 802.11(p) and 1609 Wireless Vehicular Network Researches. *68th IEEE Vehicular Technology Conference*, pp. 1–2, IEEE, 2008.

III

THESIS APPENDIX