# Correct Dynamic Service-Oriented Architectures

## Modeling and Compositional Verification with Dynamic Collaborations

Basil Becker, Holger Giese, Stefan Neumann

**Technische Berichte Nr. 29**

des Hasso-Plattner-Instituts für
Softwaresystemtechnik
an der Universität Potsdam


Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 29

Basil Becker | Holger Giese | Stefan Neumann

# Correct Dynamic Service-Oriented Architectures

## Modeling and Compositional Verification with Dynamic Collaborations

Universitätsverlag Potsdam

**Bibliografische Information der Deutschen Nationalbibliothek**
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

**Universitätsverlag Potsdam 2009**
http://info.ub.uni-potsdam.de/verlag.htm

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 4623 / Fax: 4625
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam**.**

Basil Becker, Holger Giese and Stefan Neumann
System Analysis and Modeling Group,
Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmertstr. 2-3, D-14482 Potsdam, Germany
[Basil.Becker|Holger.Giese|Stefan.Neumann]@hpi.uni-potsdam.de
March 2009



Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URL http://pub.ub.uni-potsdam.de/volltexte/2009/3047/
URN urn:nbn:de:kobv:517-opus-30473
[http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-30473]

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-940793-91-1

**Abstract**

Service-oriented modeling employs collaborations to capture the coordination of multiple roles in form of service contracts. In case of dynamic collaborations the roles may join and leave the collaboration at runtime and therefore complex structural dynamics can result, which makes it very hard to ensure their correct and safe operation. We present in this paper our approach for modeling and verifying such dynamic collaborations. Modeling is supported using a well-defined subset of UML class diagrams, behavioral rules for the structural dynamics, and UML state machines for the role behavior. To be also able to verify the resulting service-oriented systems, we extended our former results for the automated verification of systems with structural dynamics [7, 8] and developed a compositional reasoning scheme, which enables the reuse of verification results. We outline our approach using the example of autonomous vehicles that use such dynamic collaborations via ad-hoc networking to coordinate and optimize their joint behavior.

# Contents

# Chapter 1

# Introduction

The traditional approach for architectural modeling employs components with ports and interfaces to decouple the different components. The service-oriented approach in contrast employs collaborations describing the interaction of multiple roles in form of service contracts (cf. [3, 9]) and thus is able to capture complex interaction schemes between multiple roles in a more comprehensive manner than interfaces and ports which at most capture bidirectional interaction schemes.

In the service-oriented approach *orchestration* describes a collaboration with a single dedicated coordinator that enacts the collaboration between the other parties. The *choreography* interaction scheme in contrast support the free interplay of the different roles within a collaboration.

In addition, advanced applications such as distributed self-adaptive systems [22] or advanced mechatronic systems [29] require that the roles of collaborations can be assigned at run-time rather than only statically and that the number of roles and their assignment may vary over time. Therefore, *dynamic collaborations* which are not restricted to orchestration but also support choreography and where the participants and their assigned roles can change dynamically at run-time have to be supported when modeling *dynamic service-oriented architectures*.

If employed for safety-critical or high-integrity systems, proper means for verifying such systems are necessary, too. However, the automated verification of systems with structural dynamics is already a major challenge for a fixed number of roles. If dynamic collaborations with a dynamic number of roles are considered, the state space of the resulting models is often not finite and direct automated verification with model checking as proposed in [13, 5, 30, 26, 23] is therefore not applicable. On the other hand, existing approaches to address infinite state systems [4, 6] are usually too restricted to be applicable.

In [17], we presented an approach to guarantee crucial safety properties for patterns/collaborations with a fixed number of roles using model checking. The behavior of the interaction of a collaboration and its roles could be verified separately. Then, these results could be combined in a compositional manner by verifying the synchronization of each component separately exploiting that the components refine the collaboration

roles. We also developed an approach [7, 8] to address the automatic formal verification of systems with structural dynamics, as it results from rules to join and leave a collaboration, employing UML models that are extended using concepts from graph transformation systems. The technique automatically verifies inductive invariants for the potentially infinite state models to exclude unsafe situations. Also a first combination of the former two approaches to verify the coordination for one collaboration and the outlined pure structural rules have been presented in [15]. However, this former approach is limited to solutions where the collaboration is instantiated or terminated as a whole, only a small, finite number of static roles per collaboration exist, the reactive behavior of the roles itself can be fully decoupled from the structural dynamics and the collaboration does not support parameter passing.

In this paper we present an approach, which overcomes these limitations. We can enable the modeling and verification of service-oriented systems with dynamic collaborations where the roles can join or leave at runtime, the number of roles is not necessarily restricted to a small finite number, the reactive behavior of the roles itself has not be fully decoupled from the structural dynamics and the safe collaboration may also depends on the exchanged parameters between the roles.

The dynamic collaborations and service-oriented systems can be modeled by a well-defined subset of UML class diagrams, behavioral rules for the structural dynamics and UML state machines for the role behavior. For those notations we provide profound formal semantics, which are based on graph transformations. Those semantics then allows us to verify the collaborations and the service-oriented systems employing our former results for the automated verification of systems with structural dynamics. Further, we have extended these verification technique to also support the reactive behavior with unbounded integer attributes. The increased complexity of service-oriented systems with structural-dynamics and reactive behavior, which may interfere with each other, can be tackled by a compositional reasoning scheme we present in this paper, too.

The main contributions of the paper concerning modeling and verifying systems with dynamic collaborations are: (1) An *extended formally underpinned modeling approach* for systems with structural adaptation and dynamic collaborations which besides structural changes and clocks supports also integer values and parameters with simple updates. (2) An extended *checking algorithm for inductive invariants* for systems with structural adaptation and dynamic collaborations, which supports besides structural changes and clocks also integer values with simple updates. (3) A *compositional reasoning scheme* which permits to verify guaranteed properties of dynamic collaborations upfront and to *reuse* in the verification of the systems which use them that drastically reduced the verification effort and therefore makes also the verification of more complex systems with dynamic collaborations feasible.

Throughout the paper we use an application example, which is based on the RailCab[1] research project. RailCab's intention is the development of a new railway technology. The main elements of RailCab are small and autonomous shuttle. Customers

[1] http://www.railcab.de and [29]

can order these via the Internet and book for a direct connection. To reduce the shuttles' energy consumption convoys of multiple shuttles are built, which then drive in close proximity. Therefore, the shuttles in a convoy have to be coordinated, as otherwise collisions between them could be not excluded. The coordination among the Shuttles is achieved by exchanging so called braking profiles, which assign a minimum and a maximum position for a Shuttle for each point in the future [16].

In this manner the performance of such systems could be drastically improved, if ad-hoc networking and suitable real-time coordination is employed to enable a coordinated behavior of the autonomous units (cf. [24]). Related examples are automotive systems based on car-2-car communication which support the driver when approaching a crossing or concepts to build convoys of autonomous vehicles. However, due to the involved ad-hoc connections, dynamic formation building, and the real-time interaction it is hard to ensure the correctness of such coordinated autonomous units (cf. [21]).

We present first results on the verification of this case study obtained using a newly developed prototype.[2]

The paper is structured as follows: We first review the current state-of-the-art for approaches tackling the modeling and verification of service-oriented systems or systems with dynamic structural adaptation in Section 2. Then, we outline the employed modeling approach, which captures the service contract instantiation and termination based on UML in Section 3 and present the underlying formal model and the mapping to it in Section 4. The new verification algorithm is introduced in Section 5 and the new compositional reasoning scheme, which allows reusing results from the separate verification of the dynamic collaborations is outlined in Section 6. We finish the paper with a conclusion and outlook on future work.

[2]The prototype extends like [17, 7, 8] Fujaba (www.fujaba.de).

# Chapter 2

# State of the Art

Modeling using roles and focusing on collaborations rather than components is not new: Since the 1970s the OOram Software Engineering method [25] has been developed which provides a clear distinction between roles and objects and separates different collaborations in form of role models. The idea of contracts, which has been introduced in [18], also already supports a number of participants and in addition results in some *contract obligations* the classes that take over the role of the participants have to fulfill. Also a less clear historical connection between roles/collaborations and design pattern [14] exists, which is reflected today by the fact that design patterns can be modeled in UML using collaborations.

The use of collaborations for the modeling of services has been proposed by several authors (cf. [28, 9]) as well as all proposals for a UML Profile and Meta-Model for Services [3, 1]. In [28] static but hierarchic UML collaborations and the distinction between the collaboration and the collaboration use are presented. However, the authors omit the definition of the roles' behavior. An approach not using UML that overcomes this limitation is presented in [9] which uses sequence diagrams for potentially incomplete early behavior specifications. The UML Profile [3] is conceptually similar to [28]. It further extends [28] also supporting behavior specifications for the different roles.

UML class diagrams for the structure and graph transformations for the behavior modeling are also employed in [5] to model service-oriented architectures, but in contrast to our approach services are not modeled as collaborations.

We can conclude that none of the modeling concepts supports dynamic collaborations as addressed in this work.

To our best knowledge no work exists which especially addresses the problem to verify dynamic collaborations, however, a number of related approaches for the verification of service-oriented systems exist. Model checking has been employed to check business process models with varying number of active process instances. In [12], for example, standard BPEL models are enriched by resource allocation behavior to ensure the correct detection of deadlocks and safety violations for web services compositions under resource constraints. In [10] an approach dedicated to the compositional verification of middleware based software architectures is presented. The verification of

a software architecture is divided into the verification of properties, which hold for the middleware and those, which hold for the complete architecture. However the approach does not cover structural dynamics and is restricted to finite state systems.

For systems with structural dynamics like our earlier work [7] some work has been published, which does not cover dynamic collaborations to their full extent: An approach which has been successfully applied to verify service-oriented systems [5] is the one of Varró et al. It transforms visual models based on graph theory into a model-checker specific input [30]. A more direct approach is GROOVE [26] by Rensink where the checking works directly with the graphs and graph transformations. DynAlloy [13] extends Alloy [19] in such a way that changing structures can be modeled and analyzed. For operations and required properties in form of logical formulae it can be checked whether given properties are operational invariants of the system. In [21] a petri net variant is employed for the modeling and verification of some issues of an intelligent transportation system and it is suggested to use classical model checking techniques.Real-Time Maude [23], which is based on rewriting logics, is the only approach we are aware of covering structural changes as well as time. The tool supports the simulation of a single behavior of the system as well as bounded model checking of the complete state space, if it is finite. However, all these approaches do not fully cover the problem as they require an initial configuration and only support finite state systems (or systems for which an abstracted finite state model of moderate size exist).

There are only first attempts that address the verification of infinite state systems with dynamic structure: In [4] graph transformation systems are transformed into a finite structure, called Petri graph which consists of a graph and a Petri net, each of which can be analyzed with existing tools for the analysis of Petri nets. For infinite systems, the authors suggest an approximation. The approach is not appropriate for the verification of the coordination of autonomous vehicles even without time, because it requires an initial configuration and the formalism is rather restricted, e.g., rules must not delete anything. Partner graph grammars are employed in [6] to check topological properties of the platoon building. The partner abstraction is employed to compute over approximations of the set of reachable configurations using abstract interpretation. However, the supported partner graph grammars restrict not only the model but also the properties, which can be addressed a priori.

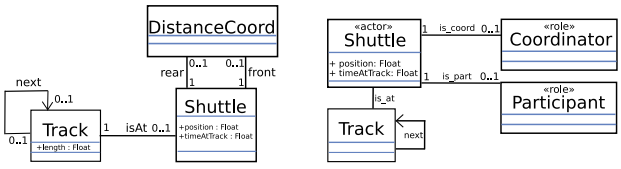
# Chapter 3

# Modeling

In this section we outline the notations we employ for modeling systems with dynamic structure and how we extend them to model collaborations and the combination of both.

## 3.1 Systems with Dynamic Structure

Systems with dynamic structure, such as the RailCab system, can be described by the means of UML class and UML object diagrams, where the UML object diagrams describe the system's states (cf. [20, 7]).

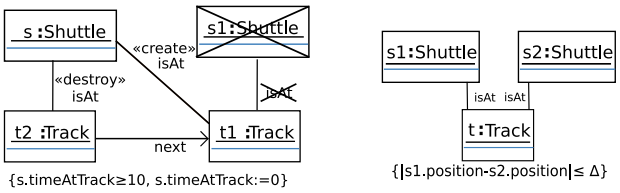



(a) Basic system (b) RailCab system

Figure 3.1: Class diagrams for the systems

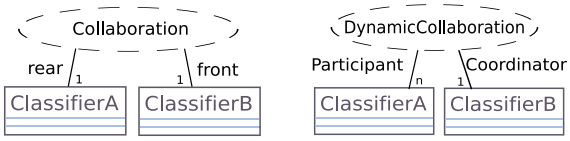
In Figure 3.1(a) the class diagram describing the elements of the application example is shown. The system consists of Shuttles, which are located at exactly one Track and Tracks that have at most one successor each. Like shown in [7] patterns/collaborations are used to coordinate the movement of the Shuttles. A Shuttle can be part of a collaboration represented through the DistanceCoord object via links realizing the rear or front association. An object of type DistanceCoord is connected to exactly one Shuttle over the rear association and to exactly one Shuttle over the front association.

The system behavior for the structural dynamics is modeled by rules in form of Story Patterns (cf. Figure 3.2(a)). A Story Pattern is an UML object diagram, whose elements are augmented with the special stereotypes ≪create≫ or ≪destroy≫ to describe the side-effects of applying the rule (cf. [20, 7]). A Story Pattern is enabled if the elements with no or the ≪destroy≫ stereotype attached could be matched. The application of a




(a) The move rule (b) Hazard collision

Figure 3.2: Hazard and move rules




(a) Simple Collaboration (b) Dynamic Collaboration

Figure 3.3: Different types of collaborations

Story Pattern deletes and creates elements w.r.t. their stereotype. We will later refer to these Story Patterns as *modification rules*.

Forbidden situations, i.e. hazards or failures, are modeled with Story Pattern without side effects. In the special case of our application example we want our system to be free of collisions (cf. Figure 3.2(b)). More details on modeling system with dynamic structure with Story Pattern could be found in [20, 7].

## 3.2 Dynamic Collaboration

Dynamic collaborations are collaborations with a varying number of participants. In contrast to the DistanceCoord collaboration described before where only a fixed number of collaborating elements exist, dynamic collaborations can consist of an arbitrary number of participating objects and the number of participants as well as the participating objects can change at runtime.

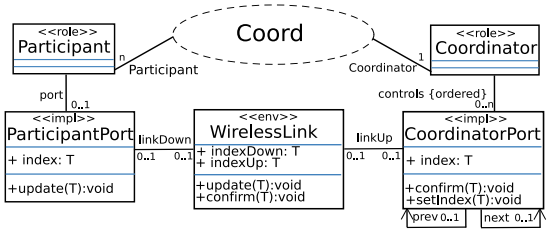



Figure 3.4: The dynamic collaboration Coord described in UML notation

This characteristic is described in form of the cardinalities in the UML collaboration diagram shown in Figure 3.3 where Collaboration consists of a fixed number of participat-

ing elements and the DynamicCollaboration can consist of an arbitrary number of Participants and a single Coordinator.

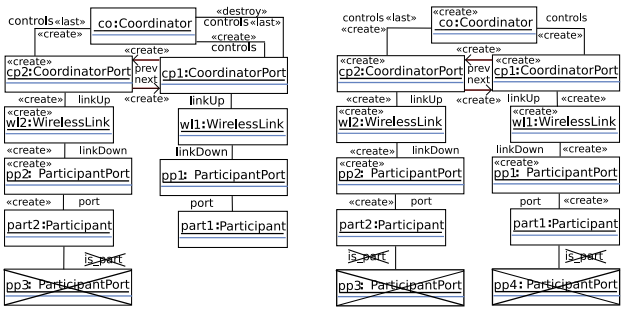
However, the rather abstract description of dynamic collaborations as depicted in Figure 3.3 (b) is not sufficient when we want to consider also the behavior of the dynamic collaboration. Therefore, to be able to describe the communication of the dynamic collaboration we in addition employ ports such as CoordinatorPort and ParticipantPort, objects representing the environment such as WirelessLinks as well as links between them encoding the communication connections as shown in Figure 3.4. Following the dynamic collaboration shown in Figure 3.4 is called Coord.

Usually when using UML classes or components a fixed number of provided or required ports are attached to components. For the dynamic collaboration Coord, however, the number of CoordinatorPorts of the Coordinator role as well as the number of ParticipantPorts and the respective Participant roles changes dynamically. Therefore, the structure of the dynamic collaboration Coord models the communication ports explicitly as separate objects.

Parts of the environment in which the dynamic collaboration is deployed may also play an important role in the behavior. If so, we represent them explicitly like in case of the WirelessLink environment object.

Coord consists of the classes Participant, ParticipantPort, WirelessLink, CoordinatorPort and Coordinator. Participant and Coordinator both have a ≪role≫ stereotype attached, which identifies them as roles within Coord. Each Participant has at most one ParticipantPort associated but a Coordinator could have an arbitrary number of CoordinatorPorts. The ports implement the communication, hence the stereotype ≪impl≫ is attached. A ParticipantPort could be connected via the WirelessLink to at most one CoordinatorPort and vice versa. CoordinatorPorts could be connected to a previous or a next CoordinatorPort over the associations prev and next. The stereotype ≪env≫ is applied to the WirelessLink to identify it as a part of the environment.

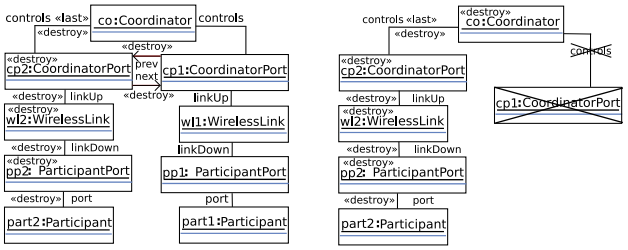
The structural dynamics for the dynamic collaboration, like the instantiation and the destruction are modeled using Story Patterns. Figure 3.5 shows the modification rules (a) join and (b) create for our dynamic collaboration Coord. These rules have to check that the given roles do not already participate in any collaboration of the same type.




(a) join rule (b) create rule

Figure 3.5: Modification rules for Coord

The leave and destroy rule is shown in Figure 3.6. The leave rule shown in Figure 3.6(a) removes the elements, which are added by the join rule shown in Figure 3.5(a). The destroy rule deletes the elements which are created by the create rule including the Coordinator, but only if there is no other CoordinatorPort connected to the Coordinator.




(a) leave rule (b) destroy rule

Figure 3.6: The leave and destroy rule for **Coord**

The reactive behavior of the dynamic collaboration is described in form of UML state machines, which are assigned to the ports and environment classes. This allows to also model the communication with a unbounded number of participants with simple state machines while assigning state machines to the roles would require to have parameterized state machines.

The state machines realize the propagation of indices between the Coordinator and the Participants via the corresponding ports and the connected WirlessLinks as follows. The Coordinator role uses the CoordinatorPort to communicate via a WirelessLink with the ParticipantPort and vice versa. During communication the CoordinatorPort propagates indices (objects of the countable and partially ordered type T) he receives to the ParticipantPort via the WirelessLink. Afterwards the ParticipantPort confirms to the CoordinatorPort also via the WirelessLink, that he has received the index. Depending on the current structural context the behavior of CoordinatorPort differs in two cases. First, the CoordinatorPort only propagates the index to a next CoordinatorPort if such a port is connected via the next association and second, if the CoordinatorPort is the first one (no other CoordinatorPort is connected via the prev association) the own index is incremented by one before it is propagated to the ParticipantPort or send to the next CoordinatorPort.

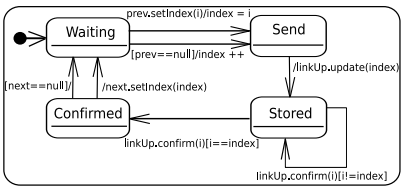



Figure 3.7: State machine **cPort**

The behavior of the CoordinatorPort is described by the state machine cPort shown in Figure 3.7. cPort starts in the state Waiting. Depending on the situation if a previous or next CoordinatorPort exists the behavior of the state machine differs. The variation of the

behavior is realized within transition guards such as in the lower transition between the states Waiting and Send. If no previous CoordinatorPort is connected via the association prev then the index of cPort can be incremented by one and the cPort changes to the state Send. If a previous CoordinatorPort exists the only possibility to change from state Waiting to Send is to receive the message setIndex over the channel prev[1] (channels are represented through links realizing associations, e.g., through the association linkUp between the classes WirelessLink and CoordinatorPort shown in Figure 3.4). In this case the index received by the message is stored. When in state Send the index of cPort is sent with the message update via the channel linkUp to the state machine of the corresponding WirelessLink and cPort changes to state Stored. In state Stored cPort waits for the message confirm over the channel linkUp. If the message confirm contains the parameter value $i$ with $i == index$ cPort changes to state Confirmed. Otherwise cPort uses the self transition of the state Stored. If a CoordinatorPort is connected over the next association the index can be sent via the message setIndex over the channel next. If such a CoordinatorPort is not connected the message is not sent. In both cases the state machine can change to the initial state Waiting.

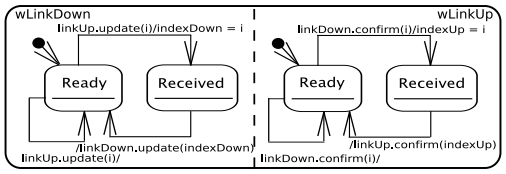



Figure 3.8: State machine of **WirelessLink**

The state machine for the WirelessLink is shown in Figure 3.8. The WirlessLink realizes behavior for sending an index from the CoordinatorPort to the ParticipantPort and vice versa. To send an index via the WirelessLink to the ParticipantPort, the CoordinatorPort sends the message update via the channel linkUp with the parameter $i$ as the index (cf. Figure 3.7).

On the left side of Figure 3.8 the state machine wLinkDown is shown which receives the index via the message update using the channel linkUp and changes from the state Ready to the state Received. With the transition from Received back to Ready this index is sent via the message update on the channel linkDown to the ParticipantPort. On the right side of Figure 3.8 the orthogonal state machine wLinkUp is shown which sends an index in the opposite direction using the message confirm. When using unreliable asynchronous communication (like in the case of a wireless link) it is possible to lose indices. This characteristic is covered by the Ready states' self transitions.

The behavior of the ParticipantPort is shown in Figure 3.9. When the state machine of the ParticipantPort has received the index by the message update in state Waiting from the WirelessLink like described above, it could receive any newly arriving index using the self transition of the state Stored. When in state Stored the last stored index could be sent

[1] Communication between different CoordinatorPorts could be realized without any communication link because all CoordinatorPorts of a collaboration are assigned to a single role and thus run on the same node.

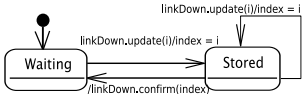
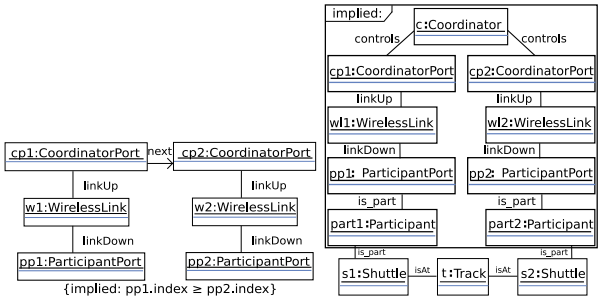



Figure 3.9: State machine **pPort**

back to the WirelessLink via the message confirm via the channel linkDown. This index is then sent to the CoordinatorPort using the state machine of the connected WirelessLink.




(a) Property **ordering** (b) Property **coordExists**

Figure 3.10: Guaranteed properties

For each dynamic collaboration we define a set of hazards, i.e. situations that should not occur, as well as guaranteed properties, i.e. implications which must hold for specific situations. While for hazards simply the forbidden situation is modeled, for a guaranteed property we describe that a condition or structure is implied. A guaranteed property named ordering for our application example is shown in Figure 3.10(a). The meaning is: The existence of the depicted structure implies that the indices, assigned to the single participants, are partially ordered.

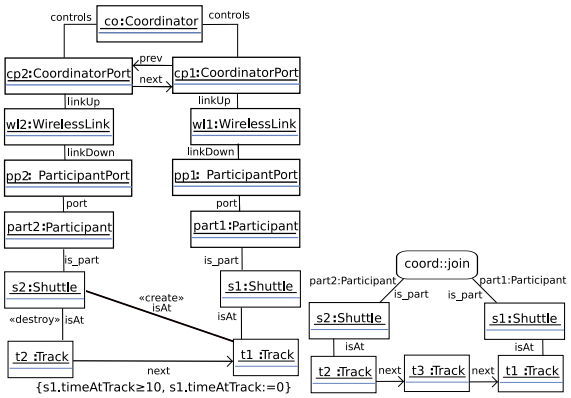
A complete set of such guaranteed properties or excluded hazards together with the modification rules which are embedded into a model when it comes to a collaboration use define the *interface* of a collaboration. As we will demonstrate later, the interface helps to reduce the complexity of the design when using collaborations and also enables their compositional verification.

## 3.3 Systems with Dynamic Collaborations

Systems with dynamic collaborations are a combination of the two concepts introduced above. The class diagrams of both concepts are combined by specifying which actor is playing which role defined in the class diagram for dynamic collaborations.

Figure 3.1(b) depicts how the classes shown for the dynamic collaboration Coord in Figure 3.4 and the classes of the dynamic system are merged together to build the combined class diagram of the RailCab system. The Shuttle class has two associations, which allows it to play the Participant or the Coordinator role.

In order to be used in the context of the merged class diagrams, the collaboration's modification rules have to be linked to the elements defined in the class diagram of the system. By construction the applied modification rules then refine the modification rules defined for Coord.




(a) Modified moveCoord rule (b) Simplified join rule

Figure 3.11: Extended move rule and simplified join rule

The system's *normal* rules could use the classes defined in the collaboration's diagram of Coord (cf. Figure 3.4). For our application example this offers the possibility to move one Shuttle at a Track where already another Shuttle is located (cf. Figure 3.11(a)), if Coord is instantiated between the two Shuttles. Then, the collaboration ensures that due to the coordination between the shuttles no collision can occur (cf. Figure 3.2(b)).

Also the guarenteed property that Coord exists when two Shuttles are at the same track could be specified in a merged diagram like shown in Figure 3.10(b).

If a modification rule is accordingly extended to the systems context, this is specified using the shorthand as depicted in Figure 3.11(b) for the join rule. The rounded rectangle references the dynamic collaboration Coord's modification rule join (cf. Figure 3.5(a)). The association is_part (cf. Figure 3.1(b)) is used to connect the Shuttles s1 and s2 to the roles of the Coord's join rule. The additional labels part1 and part2 specify that s1 and s2 are associated with the Participant roles part1 and part2 respectively.

# Chapter 4

# Formal Semantics

To enable the later described automated verification, we require that our extended UML models are equipped with a formal semantics. We therefore used graph transformation systems (GTS) [27] as underlying formal model and extend them by integer attributes and clocks.

## 4.1 Formal Model

Typed graphs (in the following we write only graphs) are used to represent the system state. A graph $G$ is a pair $(V, E)$ with $V$ a set of vertices and $E \subseteq V \times V$ a set of edges. In addition, a set of types $\mathcal{T}$ and a function $t : V \cup E \to \mathcal{T}$ assigning types to all elements is assumed. A graph $G'$ *matches* a graph $G$, if there exists an isomorphic function $iso$ that maps all elements of $G'$ to elements of $G$ of the same type. A graph pattern $P = (P^+, P^-)$ is a pair of a positive graph $P^+$ and a negative application condition (NAC) $P^-$ with $P^+ \subsetneq P^-$. $P$ *matches* a graph $G$, if there exists an isomorphic function $iso$ that maps all positive elements of $P$ ($P^+$) to elements of $G$ of the same type and no isomorphic function $iso'$ exists which extends $iso$ and maps at least one negative element of $P$ ($P^-$ but not in $P^+$) to elements of $G$ of the same type. We write $G' \lesssim_{iso} G$ resp. $G' \lesssim G$ if the specific $iso$ does not matter. If for two graphs $iso$ and $iso'$ exists such that $G' \lesssim_{iso} G$ and $G \lesssim_{iso'} G'$ holds, we write $G \approx G'$.

A complex graph pattern $P = (P^+, \{P_1^-, \ldots, P_n^-\})$ is a pair build of a graph $P^+$ and set of negative application conditions (NACs) $\{P_1^-, \ldots, P_n^-\}$ with $P^+ \subsetneq P_i^-$ for all $1 \leq i \leq n$. $P$ *matches* a graph $G$, if there exists an isomorphic function $iso$ that maps all positive elements of $P$ ($P^+$) to elements of $G$ of the same type and no isomorphic function $iso'$ exists which extends $iso$ and maps at least one NAC of $P$ ($P_1^-, \ldots, P_n^-$) completely to elements of $G$ of the same type. Note that a graph pattern $P = (P^+, P^-)$ equals a complex graph pattern $P' = (P^+, \{P_1^-, \ldots, P_n^-\})$ where $\{P_1^-, \ldots, P_n^-\} = \{P \subseteq P^- \text{—} P^+ \subsetneq P\}$.

A *graph transformation system* $S = (\mathcal{R}, prio)$ consists of a set of graph transformation rules $\mathcal{R}$, defining all possible transformations in the transformation system and a

function $prio$ which assigns priorities to each rule. The state (configuration) of $S$ is a graph $G$ typed via $t$. In our setting the types $\mathcal{T}$ and the function $t$ can be derived from the UML class diagram.

> *Example 1: For the UML class diagram for the system* Basic *depicted in Figure 3.1(a), we get the following set of types:* $\mathcal{T}_{\text{Basic}} = \{\text{Track}, \text{Shuttle}, \text{DistCoord}, \text{isAt}, \text{next}, \text{rear}, \text{front}\}$ *and a related typing function* $t_{\text{Basic}}$*. Analogously, the type set* $\mathcal{T}_{\text{Coord}}$ *and the typing function* $t_{\text{Coord}}$ *for the collaboration* Coord *are derived from the classes shown in Figure 3.4 and the type set* $\mathcal{T}_{\text{RailCab}}$ *and the typing function* $t_{\text{RailCab}}$ *are determined by the class diagram of Figure 3.1 (b) which references the one shown in Figure 3.4.*

A rule $r \in \mathcal{R}$ with $r = ((L^+, L^-), R)$ with graph pattern $(L^+, L^-)$ is *applicable* to a graph $G$ if $G$ matches $(L^+, L^-)$ and when no rule with higher priority can be applied. During the *application* of a rule $r = ((L^+, L^-), R)$ to a graph $G$, the elements that are in $L^+$ but not in $R$ are removed from $G$, and elements that are in $R$ but not in $L^+$ are added to $G$.

> *Example 2: The* move *Story Pattern depicted in Figure 3.2(a) could easily be translated into a rule* $r_{\text{move}} = ((L^+_{\text{move}}, L^-_{\text{move}}), R_{\text{move}})$ *where* $L^+_{\text{move}}$ *contains the graph induced by the vertices named* t1*,* t2 *and* s *except the edge between* s *and* t1*. The pattern* $L^-_{\text{move}}$ *contains the node* s1*.* $R_{\text{move}}$ *is similar to* $L^+_{\text{move}}$ *but the edge between* s *and* t2 *is replaced by the formerly excluded edge between* s *and* t1*. In general all elements with none or the* ≪destroy≫ *stereotype attached belong to* $L^+_{\text{move}}$*,* $L^-_{\text{move}}$ *consists of all elements of* $L^+_{\text{move}}$ *and all elements, which are crossed out.* $R_{\text{move}}$ *is built of elements without any or with the* ≪create≫ *stereotype.*

We write $G \rightarrow_r G'$ if rule $r$ can be applied to graph $G$ and the application results in graph $G'$. We write $G \rightarrow^* G'$ if $G$ can be transformed into $G'$ by a (possibly empty) sequence of rule applications. For a given graph transformation system $S = (\mathcal{R}, prio)$ and a graph $G$, the set of reachable graphs of $S$ starting from $G$ is denoted by $REACH(S, G) = \{G' \mid G \rightarrow^* G'\}$.

In case of an *attributed graph transformation systems* (AGTS) $S = (\mathcal{R}, prio)$ we in addition have vertex type specific attributes $a \in \mathcal{A}$. An attributed graph $(G, \alpha)$ consists of a graph $G$ as well as an assignment $\alpha$ which provides for each vertex $n$ and related attribute $a$ the current value as $\alpha(n, a)$ if $a$ is defined for $t(n)$.

> *Example 3: For the UML class diagram of the* Basic *system depicted in Figure 3.1(a), we get in addition to the types and typing function outlined in Example 1 the attribute set* $\mathcal{A}_{\text{Basic}} = \{\text{position}, \text{length}, \text{timeAtTrack}\}$ *for the vertex types such that for* length *is defined for all vertices of type* Track *and* position *and* timeAtTrack *are defined for all vertices of type* Shuttle*.*

Attributed graph patterns are also accordingly extended such that they can also contain Boolean constraints $\phi_P$ over the vertex attributes ($P = ((P^+, P^-), \phi_P)$). A *forbidden graph pattern* is such an attributed graph pattern, which has to be excluded by the later verification. In addition, for attributed graph rules $r = ((L, \phi), R, \mu)$ also an update $\mu$ can be used to determine the new attribute values. If $r$ matches a graph $(G, \alpha)$ the resulting attributed graph $(G', \beta)$ results from the related pure graph transformation $G \rightarrow_r G'$ for a graph isomorphism $iso$ and the attribute update $\beta$ such that $\beta(n, a) = \mu(iso(n), a)$ if defined and otherwise $\beta(n, a) = \alpha(n, a)$.

Hazards simply equal forbidden graph pattern. For mapping guaranteed properties with an implied part, we have to distinguish two cases how we can map them. If we have only a condition which is implied as the ordering in Figure 3.10(a), this can be easily mapped to a forbidden graph pattern with a NAC by simply negating the condition. The same is possible if the implied structure consists of a single instance. On the other hand if the implied structure consists of more than one instance such as for coordExists in Figure 3.10(b), we have to use a forbidden complex graph pattern to capture that it is forbidden that the whole implied structure is not present when the normal structure can be found.

> *Example 4: In case of the* hazard collision *depicted in Figure 3.2(b), we can do the encoding directly by translating the Story Pattern into a forbidden subgraph pattern* $P_{\text{collision}}$ *consisting of the graph pattern* $(P^+_{\text{collision}}, P^-_{\text{collision}})$ *with vertices set* $\{v_{s1}, v_{s2}, \dots\}$ *and a boolean property* $\phi_{\text{collision}}$ *which encodes* |s1.position - s2.position| $\leq \Delta$ *by the two inequalities* s1.position - s2.position $\leq \Delta$ *and* s2.position - s1.position $\leq \Delta$*. The resulting property, which excluded the hazard is then* $\Phi_{\text{collision}}$*. For the guaranteed property* ordering*, which is depicted in Figure 3.10(a), the specified structure implies a given condition. The shown property translates to a forbidden subgraph pattern* $F_{\text{ordering}}$ *consisting of the required graph pattern* $(P^+_{\text{ordering}}, P^-_{\text{ordering}})$ *with vertices set* $\{v_{part1}, v_{part2}, \dots\}$ *and a boolean property* $\phi_{\text{ordering}}$ *by inverting the boolean implication* $part1.index \geq part2.index$ *resulting in* $part1.index < part2.index$*. In case of a guaranteed property with a complex implied structure such as* coordExists *(depicted in Figure 3.10(b)), the required complex graph pattern results in the positive part* $P^+_{\text{coordExists}}$ *and the implied graph in the negative part* $P^-_{\text{coordExists}}$ *of a forbidden complex subgraph pattern* $F_{\text{coordExists}} = ((P^+_{\text{coordExists}}, \{P^-_{\text{coordExists}}\}, true))$*.*

While attributed graph transformation systems permit to model complex discrete models, we have to also support time-related behavior and thus in addition consider also *timed attributed graph transformation systems* (TAGTS) $S = (\mathcal{R}, \mathcal{R}_u, prio)$ with a set of urgent rules $\mathcal{R}_u \subseteq \mathcal{R}$ to denote rules which must be executed immediately when enabled. The set of attributes $\mathcal{A}$ is split into $\mathcal{A}_c$ and $\mathcal{A}_i$ for clocks and integer attributes respectively and clock resets and comparisons of clocks with constants can be used to describe the time related aspects of the behavior. A *timed graph transformation system* (TGTS) as supported in [8] is simply a TAGTS where $\mathcal{A}_i = \emptyset$.

To reflect the continuous character of time, we have, like for hybrid automata [2], two steps for a TAGTS: (1) At first, the classical discrete AGTS step results in a discrete change of the graph and attributes like for a AGTS. (2) Secondly, a time consuming continuous step as for a TGTS for $\delta > 0$ can result for a current state $(G, \alpha)$ in state $(G, \alpha \oplus \delta)$ with $(\alpha \oplus x)(n, a) = \alpha(n, a)$ for $a \in \mathcal{A}_i$ and $(\alpha \oplus x)(n, a) = \alpha(n, a) + x$ for $a \in \mathcal{A}_c$ iff no urgent rule $r$ exists which is enabled for $(G, \alpha \oplus \delta')$ and $\delta' < \delta$.[1] The inverse operation to $\oplus$ is $\ominus$.

## 4.2 Semantic Mapping

A dynamic collaboration $c$ can be straight forward mapped to a TAGTS by extending the set of types $\mathcal{T}$ and mapping of vertices and edges to types $t$ as described in Example 1 such that all types defined by the class diagram of the collaboration are included. We refer to the types and type mapping specific for the collaboration $c$ as $\mathcal{T}_c$ and $t_c$.

Using the types the rules of the collaboration $c$ can be derived from the Story Pattern and combined in a rule set $\mathcal{R}^c$, $\mathcal{R}^c_u$ are encoded in $prio_c$ as outlined in Example 2 and 4. For the priorities we use simply the guideline that all rules have the same priority $0$.

Also the behavior of CoordinatorPort, WirelessLink and ParticipantPort described through state machines (compare Section 3) is translated into GTS rules. For this purpose for each transition of a state machine the required graph structure and condition over the attributes of this graph as well as the resulting updates are derived according to the following scheme: The state of an element is encoded through an integer variable which is stored as an attribute of the element. For hierarchical state machines appropriate attributes are added to the respective classes. The created rule includes the involved classes and possibly the associations (in case when two ore more transitions of different state machines are synchronized via messages, these transitions are realized through a single rule) between them. Additionally the concerned elements need to be in the appropriate state. The update of the created GTS rules does not affect the matched structure in form of the elements and associations. The update sets the state of the involved elements to the successor state of the transitions and executes all actions of these transitions. The following example shows how two synchronous transitions are realized with one GTS rule.

> *Example 5: Two synchronous transitions between two instances of the state machine* cPort *shown in Figure 3.7 are combined to a Story Pattern like shown in Figure 4.1. Two instances of the* cPort *state machine are synchronized when the first instance is connected via the* next *association (compare Figure 3.4) to the second instance and when the first is in state* Confirmed *and the second state machine is in state* Waiting*. In this case the first instance changes from state* Confirmed *to state* Waiting *and simultaneously the second*

[1] A more detailed presentation of the semantics can be found in [8].

*instance changes from state* Waiting *to* Send *while both are synchronized via the message* setIndex*. As an additional postcondition the second state machines stores the index, which it has received through the message* setIndex *from the first state machine. The derived Story Pattern matches the situation where two elements of the type* CoordinatorPort *are connected via appropriate* next *and* prev *associations (compare Figure 3.4). For comparing and storing the states of the involved elements additional variables are added. The update of the rule simply sets the states according to the successor states of the state machines. The update of attribute valuations has been formally introduced in Section 4.1 and is realized for storing the index, which is sent via the synchronous message. The resulting Story Pattern is shown in Figure 4.1.*

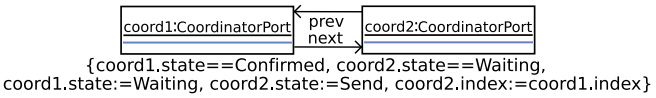



Figure 4.1: Mapping of synchronized behavior

Another example where two synchonous transitions are realized using a single GTS rule is described in Example 6 where state machines of type wLinkUp and pPort are involved.

*Example 6: Two synchronous transitions of the state machines* wLinkUp *and* pPort *are combined to a Story Pattern as shown in Figure 4.2. The first transition (shown in the lower part of* pPort *cf. Figure 3.9) is synchronized over the message* linkDown.confirm *with the second transition (shown on the top right of the state machine* wLinkUp*). The created rule matches all situation where a* WirlessLink *is connected to a* ParticipantPort *via an association* linkDown *and the found elements are in the appropriate state (state* Stored *for* pPort *and* Ready *for* wLinkUp*). For comparing and storing the states of the involved elements additional variables are added. The update of the rule simply sets the states according to the successor states of the state machines and assigns the indices (analog to the transition of the state machines). The update of attribute valuations is realized according two the same scheme like described above.*

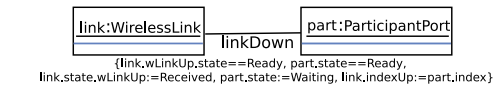



Figure 4.2: Map synchronous communication

A transition which is not synchronized with any transition of another state machine is mapped according the above shown scheme with just one element in the required

structure[2]. An example is shown in case of the state machine cPort in Figure 4.3 where the lower transition from state Waiting to state Send is not synchronized.

*Example 7: The not synchronized transition from state* Waiting *to state* Send *of the state machine* cPort *is realized through the Story Pattern shown in Figure 4.3. The Story Pattern consists of the required structural element of type* CoordinatorPort*. Additionally the transition has the guard* $prev == null$*, which requires that no other instance of type* CoordinatorPort *is connected via the* prev *association. This condition is realized in the Story Pattern in form of the shown NAC. Like described in case of the synchronous transition to be able to compare and store the state of the* CoordinatorPort *additional variables are added. As a precondition the current value of the variable* state *of the element* coord *needs to have the value* Waiting *(representing the current state). If the preconditions are fulfilled and the rule is applied the* state *variable of* coord *is set to the successor state* Send*. The side effect of increasing the index is also encoded in the additional variable* index*, which is incremented by one like shown in Figure 4.3.*

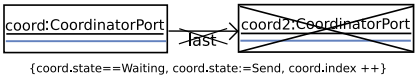



Figure 4.3: Mapping of not synchronized behavior

The properties guaranteed as well as the hazards excluded by the collaboration $c_i$ are mapped to a set of forbidden subgraph patterns $\mathcal{F}_i$ and the related property $\Phi_i$ as described in Example 2 and 4.

Then, the whole collaboration $c_i$ is captured by a TAGTS $S_i = (\mathcal{R}^i, \mathcal{R}^i_u, prio_i)$ and a property $\Phi_i$ the collaboration claim to guarantee.

*Example 8: The AGTS* $S_{\text{Coord}} = (\mathcal{R}_{\text{Coord}}, prio_{\text{Coord}})$ *for the collaboration* Coord *is accordingly built by* $\mathcal{R}_{\text{Coord}} = \{r_{\text{create}}, r_{\text{join}}, r_{\text{leave}}, r_{\text{destroy}}\}$ *and a function* $prio_{\text{Coord}}$ *such that* $prio_{\text{Coord}}(r) = 0$ *for all* $r \in \mathcal{R}$*. The property* $\Phi_{\text{Coord}}$ *is derived for* $\mathcal{F}_{\text{Coord}} = \{F_{\text{ordering}}\}$*.*

A system $s$ using dynamic collaborations can also be straight forward mapped to a TAGTS by extending the set of types $\mathcal{T}$ and mapping of vertices and edges to types $t$ such that all types defined by the class diagram are included in $\mathcal{T}_s$ and the type mapping $t_s$ work accordingly. The Story Pattern and state machines can be used to derive a related rule set $\mathcal{R}^s$ as outlined before. If urgent, a rule is in addition put into $\mathcal{R}^s_u$ and the priorities are if present simply encoded in $prio_s$. Therefore, the system is finally captured by a TAGTS $S_s = (\mathcal{R}^s, \mathcal{R}^s_u, prio_s)$.

[2]Additional elements could be included to represent forbidden structures like shown in Example 7.

We use a general scheme to determine the priorities for the rules in a TAGTS. Rules, which remove a participant from or destroy a collaboration have assigned the lowest priority. Rules, which add a participant to or create a collaboration, respectively, have the highest priority assigned. All other rules have assigned a priority in between. Of course it is possible to further differentiate between the different rules to fulfill special needs.

*Example 9: For our application example the TAGTS is defined as* $S_{\text{RailCab}} = (\mathcal{R}_{\text{RailCab}}, prio_{\text{RailCab}})$ *with rule set* $\mathcal{R}_{\text{RailCab}} = \{r^{\text{RC}}_{\text{join}}, r^{\text{RC}}_{\text{leave}}, r^{\text{RC}}_{\text{create}}, r^{\text{RC}}_{\text{destroy}}, r^{\text{RC}}_{\text{moveSimple}}, r^{\text{RC}}_{\text{move}}, \}$, *typeset* $\mathcal{T}_{\text{RailCab}} = \{\text{Shuttle}, \text{Track}, \text{Participant}, \text{Coordinator}, \text{ParticipantPort}, \text{WirelessLink}, \text{CoordinatorPort}\}$ *and priorities*

$$prio_{\text{RailCab}}(r) = \begin{cases} 0 & \textit{for } \text{leave, destroy}, \\ 1 & \textit{for } \text{move, moveSimple} \\ 2 & \textit{for } \text{create, join} \end{cases}$$

*The set of urgent rules* $\mathcal{R}_u^{\text{RailCab}}$ *consists of the rules* create *and* join*.*

With our current verification algorithm we are not able to check the existence of situations, which include statements about the position of Shuttles like shown in Figure 3.2(b). Assuming that the Shuttles are able to ensure that no collision will occur if they are on the same Track and a corresponding collaboration exists, we can reformulate the hazard shown in Figure 4.4. To be able to check whether this forbidden situation can be reached the NAC of the pattern needs to be considered. Such a complex NAC[3] like shown in Figure 4.4 could not be directly included for technical reasons using the current version of our verification tool.

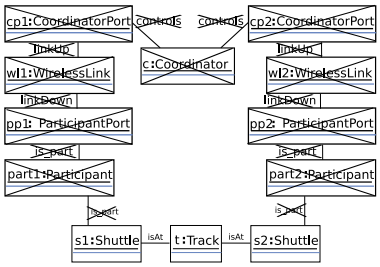


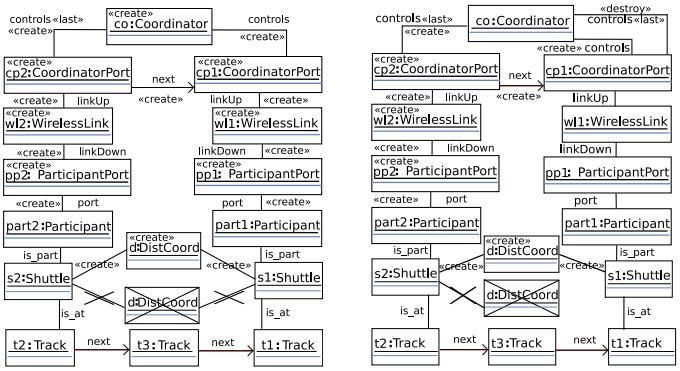

Figure 4.4: Collision hazard

To solve this technical problem a simplified representation of the NAC is used. To describe the forbidden situation that two Shuttles are on the same Track and the structure representing the corresponding collaboration does not exist (cf. Figure 4.4) a shorthand

[3]Here a complex NAC consists of more than one forbidden element in form of nodes respectively classes.

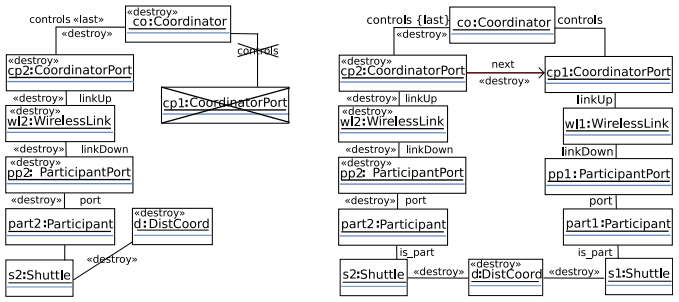
is used. Each time such a structure is created (like described by the complex NAC in negated form) an additional element of type DistCoord is created like shown in Figure 4.7.

To ensure that the DistCoord element is created each time the corresponding structure is created and that it is destroyed each time the structure is destroyed the join, leave, create and destroy rule desribed in Section 3.2 need to be extended accordingly like shown in Figure 4.5 and 4.6[4].




(a) create rule (b) join rule

Figure 4.5: Extended create and join rule for **Coord**




(a) destroy rule (b) leave rule

Figure 4.6: Extended leave and destroy rule for **Coord**

Using the DistCoord element the hazard shown in Figure 4.4 can be described more simple like shown in Figure 4.7. The NAC consists of one element of type DistCoord and the two edges connected to this element.

If such a situation exists where two Shuttles are on the same Track which are not connected to an DistCoord element could be checked by a pair of forbidden graph pattern without a complex NAC. Following we show how for the type of complex NACs, like they are used in this work such a pair of forbidden graph pattern without a complex NAC can be derived using a shorthand like in case of the above shown example for the DistCoord element.

[4]Also the join and leave rule need to be considered because they create or delete elements of the considered forbidden structure.

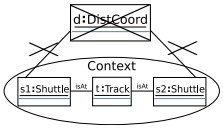



Figure 4.7: Collision hazard simple

Given a simplified hazard $P = ((P^+, P^-), \phi_P)$ we call the graph $P^+$ the hazard's *structural context* and the only node[5] existing in $P^-$ the hazard's *collaboration node*. The hazard's structural context and collaboration node are connected via a set of edges, which is given by $E_{P^-}$ (cf. Subsection 4.1)– the edges connecting the hazards positive part with its negative application condition. We can translate each simplified hazard into a set of properties, which a) ensure that the collaboration node exists (existence) and b) that the nodes contained in the structural context are all connected to the same hazard (identity).

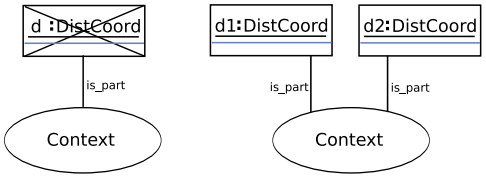
The set of properties required to check the existence of the collaboration node $c$ can be created in the following way: For each edge $e$ in the set $E_{P^-}$ we create a new attributed pattern $P_e^{ex} = ((P^+, P^-_{e,ex}), \phi_P)$, with $P^-_{e,ex}$ consisting only of $c$ and $e$. The set of all such properties created for a simplified hazard $P$ is referenced as $EX(P)$.

> *Example 10: For the simplified hazard depicted in Figure 4.7 the set $EX(P)$ contains, due to isomorphism, only one element. Figure 4.8(a) shows the contained existence property. The positive pattern $P^+$ is identical to the simplified hazard's positive pattern (cf. Figure 4.7), the negative pattern $P^-_{is\_part,ex}$ contains the collaboration node* d*, which is of type* DistCoord*, and the edge* is_part*, which is connected to* d *and* s1*. Note that the matching of pattern only relies on the pattern's structure and the occurring types, hence it is no difference whether* is_part *is connected to* s1 *or* s2*.*

The idea for checking that only one collaboration node exists for a given structural context is to iteratively build all possible patterns, where the structural context is connected to two disjoint collaboration nodes. The set $E^2 = E^- \times E^-$ contains all pairs of edges we have to investigate. For each pair $(r, s) \in E^2$ we create a pattern $P_{(r,s)} = ((P^+_{(r,s)}, \emptyset), \phi_P)$, where $P^+_{(r,s)}$ consists of $P^+$, the node $c$ and its duplicate $c'$ and the edge $r$ and $s'$. The edge $s'$ differs from $s$ only in the fact that it is not connected to $c$ but to $c'$ instead. The set of all such patterns is denoted $ID(P)$. The set $ID(P)$ can be shrunken by removing isomorphic patterns. By construction a pattern $P_{(r,s)}$ is isomorphic to $P_{(s,r)}$.

> *Example 11: The reduced set $ID(P)$ for $P$ being the hazard shown in Figure 4.7 contains two elements. One of them is sketched in Figure 4.8(b). Its*

[5]The hazard $P$ is already a simplified hazard, thus the hazard's negative pattern contains exact one node only.




(a) Existence (b) Identity

Figure 4.8: Derived Hazards

*positive part contains all elements of $P$'s positive part and two collaboration nodes* d1*,* d2 *– both of type* DistCoord *– and two edges, which are both of type* is_part*. The edges connect* d1 *with* s1 *and* d2 *with* s2 *respectively. Conforming to the construction rule given above, the pattern's negative part is empty. The second pattern contained in $ID(P)$ contains the same elements but the edge formerly connecting* d2 *and* s2 *connects* d2 *and* s1 *now.*

Generally, instead of directly verifying that the property defined by a simplified hazard holds, we use an indirection step and verify instead that the properties defined by the sets $ID(P)$ and $EX(P)$ hold.

# Chapter 5

# Verification

We extend in this section our former results [7, 8] for automated verification of GTS or TGTS models towards the presented formal model of timed attributed graph transformation systems (TAGTS).

## 5.1 Foundations

The algorithm presented in [7] permits to check a property $\Phi$ defined by a set of forbidden graph pattern $\mathcal{F}$ for a GTS $S$. We have demonstrated in [7] that this problem can be tackled algorithmically with an explicit as well as symbolic algorithm.

In [8], we extended the results of [7] to also cover TGTS where vertices can have real value clocks to describe time-related behavior using linear inequalities of real variables to encode clocks.

However, the solution developed so far does not support TAGTS, which are required to encode complex state dependent behavior, which is present in the coordination protocols and systems with clocks *and* unbounded integer attributes.

The basic idea to approach also the checking for TAGTS is to extend the timed case by also supporting attributes and their updates.

A set of forbidden subgraph patterns $\mathcal{F} = \{(F_1, \psi_1), \ldots, (F_n, \psi_n)\}$, which represent configurations that are unsafe or contradict guaranteed properties and thus have to be excluded are used to define a property $\Phi$. This property holds for a graph $G$ and assignment $\alpha$, denoted by $(G, \alpha) \models \Phi$, iff $(G, \alpha)$ matches none of the graph patterns in $\mathcal{F}$. We call $(G, \alpha)$ a *witness* for the property $\neg\Phi$ if $(G, \alpha)$ in contrast matches a forbidden graph pattern $(F_i, \psi_i) \in \mathcal{F}$.

The property $\Phi$ is an *operational invariant* of a TAGTS $S = (\mathcal{R}, \mathcal{R}_u, prio)$ iff for a given initial graph $(G^0, \alpha^0)$ for all $(G, \alpha) \in \mathsf{REACH}(S, (G^0, \alpha^0))$ holds $(G, \alpha) \models \Phi$ (cf. [11]). However, as TAGTS are too expressive to check such operational invariants, we instead tackle the problem whether the property $\Phi$ is an *inductive invariant*. This is the case if for all graphs $(G, \alpha)$ and for all rules $r \in \mathcal{R}$ holds that $(G, \alpha) \models \Phi \ \wedge \ (G, \alpha) \rightarrow_r (G', \alpha')$ implies $(G', \alpha') \models \Phi$. If we have an inductive invariant and the initial graph $(G^0, \alpha^0)$ fulfills

the property, then $\Phi$ is also an *operational invariant* as inductive invariants are stronger than their operational counterparts.

We can similar to [7, 8] formulate the definition of an *inductive invariant* for TAGTS in a falsifiable form: a property $\Phi$ with forbidden subgraph patterns $\{(F_1, \psi_1), \ldots, (F_n, \psi_n)\}$ is an inductive invariant of a TAGTS $S = (\mathcal{R}, \mathcal{R}_u, prio)$ if and only if there exists no pair $((G, \alpha), r)$ of an attributed graph $(G, \alpha)$ and an attributed rule $r \in \mathcal{R}$ such that $(G, \alpha) \models \Phi$, $(G, \alpha) \rightarrow_r (G', \beta)$, and $(G', \beta) \not\models \Phi$. Such a pair $((G, \alpha), r)$, which witnesses the violation of property $\Phi$ by rule $r$ is then a *counterexample*.

As explained in detail in [7], we can exploit the fact that the application of a rule in a TAGTS can only have a local effect to verify whether a counterexample exists. A counterexample $((G, \alpha), r)$ can only exist when the local modification of $(G, \alpha)$ by rule $r$ is necessarily responsible for transforming the correct graph $(G, \alpha)$ into a graph that violates the property. As we can represent infinite many possible counterexamples by an only finite set of representative patterns $\Theta((F_i, \psi_i), R_l, \mu_l)$ of graph patterns $P'$ that are combinations of a RHS $R_l$ of a rule $r_l = ((L_l, \phi_l), R_l, \mu_l)$ and a forbidden graph pattern $(F_i, \psi_i) \in \mathcal{F}$ (cf. [7]), we can check that no counterexample exists (and $\Phi$ is thus an inductive invariant) only considering this finite set.

As depicted in Figure 5.1 we have to check for any graph pattern $(P', \phi_{P'}) \in \Theta((F_i, \psi_i), R_l, \mu_l)$ for some $(F_i, \psi_i) \in \mathcal{F}$ and $r_l \in \mathcal{R}$ whether the pair $((P, \phi_P), r_l)$ with $(P, \phi_P)$ defined by $(P, \phi_P) \rightarrow_r \rightarrow_\delta (P', \phi_{P'})$ is a counterexample for $\Phi_\mathcal{F}$ or not as follows:

1. Check that the rule $r_l$ can be applied to the attributed graph pattern $(P, \phi_P)$ and that $(P, \phi_P) \rightarrow_{r_l} \rightarrow_\delta (P', \phi_{P'})$ (this implies that no $r_k \in \mathcal{R}_u \setminus \{r_l\}$ exists with $prio(r_k) > prio(r_l)$ that matches $(P, \phi_P)$ and that for all $x \leq \delta$ holds that $(P', \phi_{P'} \ominus x)$ is matched by no $r_m \in \mathcal{R}_u$, due to the definition of rule application).

2. Check for the attributed graph pattern $(P, \phi_P)$ that $(P, \phi_P) \models \Phi$ as otherwise $(P, \phi_P)$ would be already invalid.

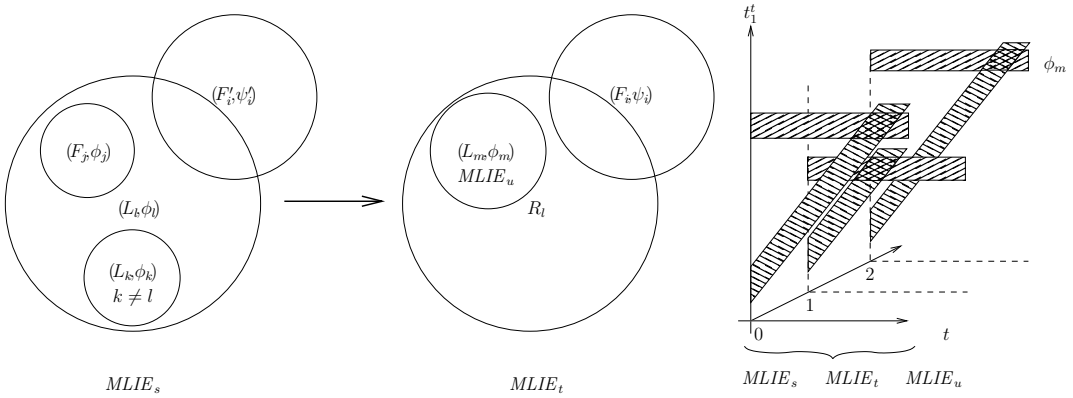



Figure 5.1: Schema to check a potential counterexample $((P, \phi_P), r_l)$ with resulting graph pattern $(P', \phi_{P'})$ that is a combination of a RHS $R_l$ of a rule $r_l$ and a forbidden graph pattern $(F_i, \psi_i) \in \mathcal{F}$ in the timed case

To do this check we combine the former purely structural checks with a system of mixed linear inequalities[1] for integer and clock variables. The rule $r_l$ could only be applied if the valuation of the integer variables and clocks in the source pattern $(P, \phi_P)$ satisfies the rule's constraint. For urgent rules with higher priority ($(L_k, \phi_k)$) as well as forbidden attributed graph patterns ($(F_j, \psi_j)$) holds in the case they also contain constraints that a match found in the source graph pattern does not directly invalidate the counterexample but rather restrict the possible clock and attribute values. We derive a system of mixed linear inequalities $MLIE_s$, which consist of constraints of integer variables and clocks, to encode which valuations are not excluded either by urgent higher priority rules or forbidden graph patterns combining the conditions $iso(\psi_j)$ for all matches $iso$ of $F_j$ in $(P, \phi_P)$ and $iso'(\phi_k)$ for all matches $iso'$ of $L_k$ in $(P, \phi_P)$.

In a TAGTS it is not required (although it is not forbidden) that $(F_i, \psi_i)$ exists in the target pattern immediately after the graph rule has been applied but after a time step of length $\delta$ with $\delta \geq 0$ it has to exists in order for the pair $(P, \phi_P) \rightarrow_{r_l} \rightarrow_{\delta} (P', \phi_{P'})$ being a witness against the system's correctness. Further, clocks and attribute values could be updated by the rule application. Considering both issues we can translate the constraints $\phi_i$ into a system of mixed linear inequalities $MLIE_t$ by substituting each occurrence of variable $a_i \in \mathcal{A}_i$ with $u_i \in \mathbb{N}$ if $a_i$ is updated to $u_i$ and each occurrence of clock $a_j \in \mathcal{A}_c$ with $a_j + t$ or in case of an update with $u_j + t$ respectively. The additionally introduced variable $t$ represents the time passed and is constraint by $t \geq 0$.

The special variable $t$ is further restricted by an upper bound, which is defined through a system of mixed linear inequalities $MLIE_u$. These upper bounds could be derived from the conditions of each embedded urgent rule. Assuming the condition $\phi_r$ is an urgent rule's condition we solve it for the special variable $t$. The result of this operation is an interval of points in time when the urgent rule $r$ is activated. Due to the fact that urgent rules have to be implied as soon as they are applicable, we only have to consider the lower bounds of this interval. These lower bounds will be used as the upper bound for the special variable $t$ in $MLIE_u$. Therefore, we only have a counterexample if a solution for the combination of $MLIE_s$, $MLIE_t$ and $MLIE_u$ can be found.[2] We write $check_{mixed}(S, \Phi)$ to denote that the check did not find a counterexample for a TAGTS $S$ and property $\Phi$.

In case of complex NACs as employed in $\Phi_{\text{coordExists}}$ (resp. $\mathcal{F}_{\text{coordExists}}$ in Figure 3.10(b)), we have to derive an extended GTS to check them with the existing algorithm. The extended GTS employs additional vertices and some related adjustments of the rules to encode the occurrence of the complex NACs such that it can be checked.

[1] A system of mixed linear inequalities is a system of inequalities, which are defined over continuous and integer variables

[2] To be able to use a constraint solver for the verification algorithm, we restrict the guards and updates to those which are formulated as linear polynomial over the objects' attributes and allow only constant multiplicities of attributes.

## 5.2 Application

This new verification technique now allows us to verify in principle the correctness of our application example, consisting of the rules move (cf. Figure 3.2(a)), moveCoord (cf. Figure 3.2(a)), $\text{join}_{\text{RailCab}}$ (cf. Figure 3.11), $\text{create}_{\text{RailCab}}$, $\text{leave}_{\text{RailCab}}$, $\text{destroy}_{\text{RailCab}}$ and the rules describing the behavior of the collaboration's automatons. The checked Property $\Phi$ is defined by the conjunction of the two guaranteed properties ordering (cf. Figure 3.10(a)) and coordExists (cf. Figure 3.10(b)). The urgent transitions in this system are the rules join and create, which both have a priority of 2. The rules leave and destroy have a priority of 0 all other rules have priority 1. The verification of this system did not work directly and we had to interrupt it after 4 hours as the machine started heavily swapping. The characteristic of this check and a comparison to later introduced compositional checks can be found at the end of Section 6.

# Chapter 6

# Compositional Verification

We will present in this section a compositional reasoning scheme for the verification of systems that use dynamic collaborations that exploits the interfaces of the dynamic collaborations. Beforehand, we discuss *abstraction* as a prerequisite.

## 6.1 Simulation

If we compare GTS of different level of abstraction or detail, the notion of a restriction | for attributed graphs concerning the visible details is important. Examples are: Restricting an attributed graph $(G, \alpha)$ to the types $\mathcal{T}_2$ of a GTS $S_2$ denoted by $(G, \alpha)|_{\mathcal{T}_2}$. Another option is to restrict an attributed graph $(G, \alpha)$ to a specific subset of attributes or clocks denoted by $(G, \alpha)|_{\mathcal{A}}$. We further use $\square$ as a placeholder for a specific restriction.

Given a restriction, we can compare two GTS of different level of abstraction or detail:

**Definition 1** *For two TAGTS* $S_1 = (\mathcal{R}^1, \mathcal{R}^1_u, prio_1)$ *and* $S_2 = (\mathcal{R}^2, \mathcal{R}^2_u, prio_2)$ *and a restriction* $\square$ *for* $S_1$ *holds that* $S_1$ *is a* simulation *of* $S_2$ *for the restriction* $\square$ *(*$S_1 \precsim_\square S_2$*) iff an* $\Omega \subseteq \{(G, G') | G, G' \in \mathcal{G} \wedge G|_\square \approx G'\}$ *exists such that* $(G_1, G_2) \in \Omega$ *implies* $G_1|_\square \approx G_2$ *and*

$$\forall G_1' : G_1 \rightarrow G_1' \;\; \exists G_2' : G_2 \rightarrow G_2' \wedge (G_1', G_2') \in \Omega \tag{6.1}$$

Simulation preserves safety properties, which are not affected by the restriction. Therefore, it is sufficient to establish a required safety property for a more abstract GTS, which is simulated by a more detailed GTS to ensure that the safety property holds for the detailed GTS. However, checking simulation via equation 6.1 is not feasible for infinite state models like GTS. Therefore, we will exploit that in specific cases simulation can be guaranteed by construction due to a refinement relation between the rule sets.

We can at first observe that the rules, which result from embedding a collaboration result in certain form of rule set refinement.

**Definition 2** *A set of rules* $\mathcal{R}^1$ refines *another set of rules* $\mathcal{R}^2$ *for a given restriction* $\square$ *iff it exists an* $\Omega \subseteq \mathcal{R}^1 \times \mathcal{R}^2$ *such that*

$$\begin{aligned}
\forall r_1 \in \mathcal{R}^1 \exists (r_1, r_2) \in \Omega : \quad & (r_1|_\square = r_2 \\
\wedge \quad & (r_2 \in \mathcal{R}^2_u \Rightarrow r_1 \in \mathcal{R}^1_u) \\
\wedge \quad & (prio(r_2) > 0 \vee r_2 \in \mathcal{R}^2_u) \Rightarrow r_1 = r_2) \\
\wedge & \\
\forall (r_1, r_2), (r'_1, r'_2) \in \Omega : \quad & (prio(r_1) > prio(r'_1) \Rightarrow prio(r_2) > prio(r'_2)) \\
\wedge \quad & (prio(r_1) < prio(r'_1) \Rightarrow prio(r_2) < prio(r'_2)) \qquad (6.2)
\end{aligned}$$

Such a system correctly use a collaboration only when it includes the types of the collaboration ($\mathcal{T}_c \subseteq \mathcal{T}_s$) and refines the rules of $\mathcal{R}^c$ correctly as outlined in Definition 2. Due to the fact that the class diagram for the system imports the types of the collaboration but is otherwise disjoint and that rules in $\mathcal{T}^s$ can only contain vertex types from $c$ if derived from a rule in $\mathcal{T}^c$ guarantees that both conditions are fulfilled by definition.

> *Example 12: The refinement between the TAGTS* $S_{\text{RailCab}}$ *and the AGTS* $S_{\text{Coord}}$ *becomes the most obvious by a comparison between Figure 3.5 and Figure 3.11. The Story Pattern used in the collaboration use (cf. Figure 3.11) adds additional elements to the precondition, but all elements added belong to types defined in the class diagram in Figure 3.1(a). Therefore each time the Story Pattern is enabled the Story Pattern in Figure 3.5 is also applicable.*

We can now show that the refinement for the rule sets from Definition 2 implies simulation.

**Lemma 1** *For two TAGTS* $S_1$ *and* $S_2$ *where* $S_1$ *refines the rule set of* $S_2$ *concerning the type set* $\mathcal{T}_1$ *of* $S_1$ *holds* $S_1 \precsim_{\mathcal{T}_1} S_2$.

***Proof:*** *(sketch) The refinement of the rule sets guarantees that for all start graphs* $G_1$ *of* $S_1$ *and steps* $r_1$ *in* $S_1$ *and* $G_2$ *defined as* $G_1|_{\mathcal{T}_S^1}$ *holds that the refined rule* $r_2$ *in* $G_2$ *can also be applied not taking preemption due to the priority or urgency into account. For the resulting graphs* $G'_1$ *and* $G'_2$ *with* $G_1 \rightarrow_{r_1} G'_1$ *and* $G_2 \rightarrow_{r_2} G'_2$ *holds* $G'_2 = G'_1|_{\mathcal{T}_S^1}$. *Concerning the preemption and urgency, we can further conclude that condition 6.2 ensures that if* $r_2$ *can be blocked due to either a rule with higher priority or urgency, this is still the case for* $r_1$ *and a related rule. Therefore, a not empty* $\Omega$ *exists for which the condition 6.1 is obviously fulfilled.*

## 6.2 Application

Exploiting the above results, we can for example check the guaranteed property of the Coord collaboration embedded into the RailCab system abstracting from the clock values of the RailCab system. Due to the reduced complexity the checking only requires slightly

more than 30 min and thus is much faster than the check of the complete RailCab in the former section which took more than 4.5 hours.

These result show that abstraction can considerable reduce the required time, but when the integer variables have to be checked together with a large number of complex rules, the scalability of the verification is still not given.

However, in the above sketched cases the abstractions have to be invented ad hoc and can not be derived from the models directly. In the next section we will demonstrate that for the presented modeling approach we can exploit the decomposition into dynamic collaborations and an embedding system to also decompose the verification steps and derive abstractions automatically.

## 6.3 Compositional Reasoning Scheme

We further developed a compositional verification scheme that allows us to check intermediate results for dynamic collaboration $c_1, \ldots, c_n$ and their TAGTSs $S_1, \ldots, S_n$ as well as the system and later compose them to provide the required guarantees for the overall system $s$ and its TAGTS $S_s$. This composition is based on the proper refinement of the collaborations $c_i$ by the systems $S_s$, which is guaranteed for the presented modeling approach by construction:

(1) In a first step, we use *independent checks*, which consider each dynamic collaboration $c_1, \ldots, c_n$ on its own. We employ $check(S_i, \Phi_i)$ to directly verify the properties $\Phi_i$ for the dynamic collaborations $c_i$.

If $\Phi_i$ does not refer to the clocks or integer variables, we can use an *abstraction* of the TAGTS $S_i$ in form of an TGTS or AGTS $S_i^a$ by simply erasing all attributed conditions and updates which refer to integer variables resp. clocks such that by construction the rules of $S_i$ refine those of $S_i^a$. We then can conclude that the ATGTS $S_i$ simulates $S_i^a$ using Lemma 1 and can then check more efficiently for $S_i^a$ with the check ($check(S_i^a, \Phi_i)$ that a hazard does not occur and then transfer this result to the ATGTS $S_i$. Note that such a check is a sufficient but necessary condition.

*Example 13: In our application example we use the $check$ to ensure that the collaboration's reactive behavior $S_{\text{Coord}}$ does not violate the guaranteed property $\Phi_{\text{ordering}}$ (cf. Figure 3.10(a)). However, enacting the check yields that the defined invariant is not sufficient to guarantee the system's safety.*

*Figure 6.1 shows the counterexample found by our algorithm. The graph at the top is one possible situation before a rule application and the graph at the bottom is the situation resulting from the rule application. The involved rule encodes a synchronous communication between* ParticipantPort *and* WirelessLink*. In the detected case the rule can write any unconstrained value to the* ParticipantPort*'s* index *attribute.*

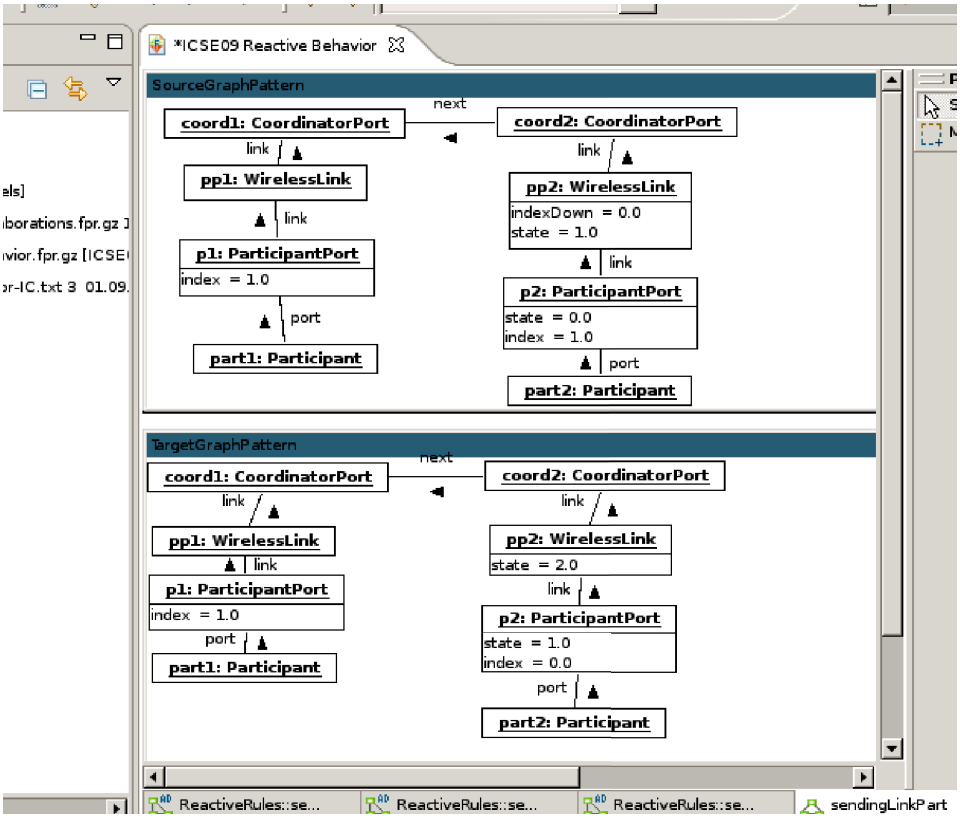
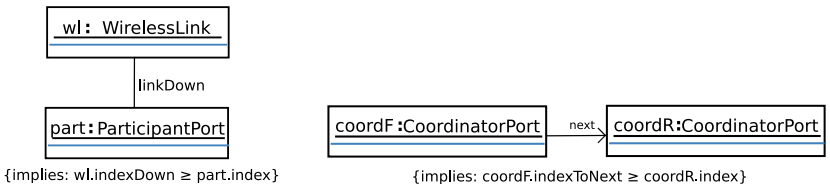



Figure 6.1: Screenshot of the counterexample




(a) Covering linkDown (b) Covering next

Figure 6.2: Additional guaranteed properties

*The provided counterexample helps us to further strengthen the guaranteed property. Repeatedly applying our check for refined versions we finally can find out that the whole data path where the index variables are propagated have to be properly covered.*

*To fix this, we in addition require that* pp1.index $\leq$ w1.indexDown *and* pp2.index $\leq$ w2.indexDown, w1.indexDown $\leq$ cp1.index *and* w2.indexDown $\leq$ cp2.index *as well as* cp1.index $\leq$ cp2.index *holds. The first and second case can be encoded together via an additional guaranteed property as depicted in Figure 6.2(a). The third one is handled by a similar rule. Finally, the correctly strengthened*

*property has been found and can be checked successfully. For all checks holds that the time required for it was $\leq 1$ sec.*

(2) Secondly, we *reuse* the checks for the collaborations by exploiting the fact that the rules in the TAGTS $S_s$ of the system using the collaborations $S_i$ refine the rules by construction. Therefore, we can conclude that $S_s$ simulates $S_i$ concerning the elements of $S_i$ ($S_s \precsim_{\mathcal{T}_i} S_i$) using Lemma 1 and can thus transfer the guaranteed property $\Phi_i$ checked for the collaboration to the system.

*Example 14: Applied to our application example, the guaranteed property $\Phi_{\text{ordering}}$ of the collaboration $S_{\text{Coord}}$ can be transferred to the systems $S_{\text{RailCab}}$.*

(3) If we require an *additional property* $\Phi_s^a$ of the detailed system itself, we often cannot check these properties directly for the TAGTS $S_s$ due to the resulting checking complexity (cf. Section 5). However, we can reuse the proven properties of the collaborations and embed the related GTS, which does not take the integer variables and clocks into account instead of the TAGTS. Using such an *abstraction* of the TAGTS $S_s$ in form of an GTS $S_s^a$ that ensures simulation, we can then check for $S_s^a$ whether $\Phi_s^a$ holds and then transfer this result to the TAGTS $S_s$.

*Example 15: We have derived a TAGTS $S^a_{\text{RailCab}}$ from the TAGTS $S_{\text{RailCab}}$ embedding for the collaboration* Coord *the abstract GTS $S^a_{\text{Coord}}$ rather than the concrete AGTS $S_{\text{Coord}}$. For this TAGTS $S^a_{\text{RailCab}}$ we then use $check$ to verify the property $\Phi_{\text{coordExists}}$ depicted in Figure 3.10(b). After only $\approx 64$ sec. the check reports that $\Phi_{\text{coordExists}}$ holds. The check therefore showed that the system is safe and excludes that shuttles nearby are not connected via the* Coord *collaboration.*

(4) Finally, we can *combine* the knowledge gathered from the check of the collaborations with the knowledge about the specific conditions occurrence of hazards for the TAGTS $S_s$ to prove the correctness and safety of the system using the capabilities and guarantees of the collaborations.

**Theorem 1** *Given a system $s$ with TAGTS $S_s$, a required property $\Psi$ for $s$, collaborations $c_1, \ldots, c_n$ with TAGTS $S_1, \ldots, S_m$ used by $s$, abstractions $S_1^a, \ldots, S_m^a$ for the collaborations $c_1, \ldots, c_n$, and $S_s^a$ the TAGTS which results when embedding $S_1^a, \ldots, S_m^a$ rather than $S_1, \ldots, S_m$ into $s$ it holds, that if a related property $\Phi_s^a$ for $S_s^a$ exists such that $(\wedge_{1 \leq i \leq n} \Phi_i) \wedge \Phi_s^a \Rightarrow \Psi$ we have:*

$$(((S_1 \models \Phi_1) \wedge \cdots \wedge (S_n \models \Phi_n)) \wedge (S_s^a \models \Phi_s^a)) \;\Rightarrow\; S_s \models \Psi. \tag{6.3}$$

***Proof:*** *(sketch) The correct usage of the dynamic collaborations $c_1, \ldots, c_n$ implies that $S_s \precsim_{\mathcal{T}_i} S_i$ and thus the precondition $S_i \models \Phi_i$ can be transferred to $S_s$ such that $S_s \models \Phi_i$ holds using Lemma 1. From $S_s^a \models \Phi_s^a$ and $S_s \precsim_{\square} S_s^a$ analogously follows $S_s \models \Phi_s^a$ using again Lemma 1. Finally, combining these findings with $(\wedge_{1 \leq i \leq n} \Phi_i) \wedge \Phi_s^a \Rightarrow \Psi$ we can conclude that $S_s \models \Psi$ holds and condition 6.3 is fulfilled.*

When exploiting the results of Theorem 1, we can at first *reuse* the properties $\Phi_i$ guaranteed by the dynamic collaboration $c_i$. If the combination is not sufficient to prove $\Psi$, we can use the abstractions $S_s^a$ and property $\Phi_s^a$, which together with the guarantees of the dynamic collaborations imply $\Psi$.

## 6.4 Application

*Example 16: As we have shown in Example 15 the collaboration is instantiated and participants are added when required in the* RailCab *system (*$S^a_{\text{RailCab}} \models \Phi_{\text{coordExists}}$*) and the modeled collaboration* Coord *fulfills the guaranteed property (*$S_{\text{Coord}} \models \Phi_{\text{ordering}}$*). Therefore, we can conclude that (1) anytime two shuttles are near each other they are driving together in the same convoy coordinated by an instantiation of the* Coord *collaboration. (2) As outlined in Section 3.3 in the concrete instantiation instead of the integer values indexed driving profiles are exchanged and thus the property* $\Phi_{\text{ordering}}$ *guarantees that the driving profile of the follower shuttle has a lower or equal index than the shuttle in the front. (3) An Additional check for the mechanical models (cf. [16]) guarantees that the driving profiles indices guarantee* |s1.position - s2.position| $> \Delta'$ *as long as the following shuttle has a lower or equal index than the shuttle in front of it. Therefore we can for* $\Delta' > \Delta$ *exclude* |s1.position - s2.position| $\leq \Delta$*, the condition for the hazard* collision *for two shuttles. Therefore, we can finally guarantee* $\Phi_{\text{collision}}$ *and thus exclude collisions.*

## 6.5 Comparison

Table 6.1: Characteristics of the checks

| characteristics | #rules | #inv | #inv' | $\lvert R\rvert$ | $\lvert F\rvert$ |
|---|---|---|---|---|---|
| $S_{\text{Coord}} \models \Phi_{\text{ordering}}$ | 7 | 7 | 8 | 2 | 3 |
| $S^a{}_{\text{RailCab}} \models \Phi_{\text{coordExists}}$ | 6 | 2 | 2 | 17 | 5 |
| $S_{\text{RailCab}} \models \Phi_{\text{coordExists} \wedge \text{ordering}}$ | 13 | 9 | 11 | 17 | 10 |

Table 6.1 (a) contains some structural characteristics for the checked systems. The columns stand for the number of rules, number of forbidden subgraph pattern, number of cardinality checks, number of nodes in the biggest rule, and the number of nodes in the biggest forbidden subgraph.

Performance measurements such as the overall checking time, the number of subgraphs generated, the number of GTS that entered the structural check and last the number of times a system of linear inequalities had to be solved is presented in Table 6.2 (b).

Table 6.2: Computation times and number of checks

| check | time | #graphs | GTS | CPLEX |
|---|---|---|---|---|
| $S_{\text{Coord}} \models \Phi_{\text{ordering}}$ | $<$1s | 420 | 213 | 51 |
| $S^a{}_{\text{RailCab}} \models \Phi_{\text{coordExists}}$ | $\approx$64s | 12245807 | 618 | 31 |
| $S_{\text{RailCab}} \models \Phi_{\text{coordExists} \wedge \text{ordering}}$ | $>$4h | $>$303060034 | $>$900 | $>$109 |

The presented times demonstrate that the addressed problems can be tackled using the compositional approach, while checking of the complete system requires too much time. A brute force attempt such as model checking which consider the reachable state space must obviously also fail due to the infinite number of graphs as well as infinite many integer assignments which can result for the model (an example for this has been presented in [7] for the case of GTS without time and without integer variables).

The benefits of our compositional scheme are twofold: At first we *reuse* proven collaborations and thus the checks for the guaranteed properties of the dynamic collaboration have to be done only once. The use of a collaboration ensures, due to the guaranteed refinement of the rules, that the proven guarantees can be taken for granted in the usage context. Secondly, the compositional approach results in a dramatically improved *efficiency* for the checking. The checks for the guaranteed properties of the dynamic collaboration can be done on a much smaller TAGTS. While the lower number of rules helps to avoid checking several unnecessary combinations, the resulting smaller rules ensure that the combinatorial explosion when considering all possible combinations of a rule and forbidden graph pattern is less dramatic. The more relevant improvement, however, can be achieved when checking properties, which must be guaranteed for the system. Due to the combination of rules from the collaboration and the embedding context (e.g., Figure 3.11(a)) the size of the rules increases considerably. By embedding the smaller and simpler GTS rules for the collaborations rather than the detailed one, the size and complexity of the resulting rules can be considerably reduced and instead of the detailed rules the guaranteed properties are exploited to proof crucial system properties. This further ensures that as long as the *interface* of the collaborations remains the same, the verification remains valid even though the details of the rules may be altered.

# Chapter 7

# Conclusion & Future Work

In this paper we presented our approach to model service-oriented systems with dynamic collaborations and outlined how crucial properties of the dynamic collaborations and systems employing them can be verified reusing the verification of the dynamic collaborations. Besides the structural dynamics the approach also supports integer variables and time-related behavior. The proposed compositional verification schemes results in a dramatic reduction of the required verification efforts by allowing to verify the collaboration independent from the embedding context and reuse this verification results for the checking of the embedding system.

As future work it is planned to develop comprehensive tool support for all the outlined modeling and verification steps and applying the approach in related domains such as automotive systems as well as other potential application areas such as high-integrity service-oriented systems in the business domain. Further we aim at developing native support for complex NACs.

# Bibliography


[1] *UML Profile and Metamodel for Services - for Heterogeneous Architectures (UPMS-HA)*, June 2007. http://www.omg.org/cgi-bin/doc?ad/2007-06-02.

[2] R. Alur, C. Coucoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid Automata: an algorithmic approach to the specification and verification of hybrid systems. In R. Grossmann, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems I*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer Verlag, 1993.

[3] J. Amsden, P. Rivett, K. Henk, F. Cummins, J. Mukerji, A. Lonjon, C. Casanave, and I. Badr. *UML Profile and Metamodel for Services*, June 2007. http://www.omg.org/docs/ad/07-06-03.pdf.

[4] P. Baldan, A. Corradini, and B. König. A static analysis technique for graph transformation systems. In *Proc. CONCUR*, volume 2154 of *LNCS*, pages 381–395. Springer, 2001.

[5] L. Baresi, R. Heckel, S. Thöne, and D. Varró. Modeling and validation of service-oriented architectures: Application vs. style. In *Proc. ESEC/FSE*, pages 68–77. ACM, 2003.

[6] J. Bauer and R. Wilhelm. Static Analysis of Dynamic Communication Systems by Partner Abstraction. In *Proc. of the 14th International Symposium, SAS*, volume 4634 of *LNCS*, pages 249–264. Springer Berlin / Heidelberg, 2007.

[7] B. Becker, D. Beyer, H. Giese, F. Klein, and D. Schilling. Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation. In *Proc. ICSE*. ACM, 2006.

[8] B. Becker and H. Giese. On Safe Service-Oriented Real-Time Coordination for Autonomous Vehicles. In *Proc. ISORC*. IEEE Computer Society Press, 2008.

[9] M. Broy, I. H. Krüger, and M. Meisinger. A formal model of services. *ACM Trans. Softw. Eng. Methodol.*, 16(1):5, 2007.

[10] M. Caporuscio, P. Inverardi, and P. Pelliccione. Compositional Verification of Middleware-Based Software Architecture Descriptions. In *Proc. ICSE*, pages 221–230, 2004.

[11] M. Charpentier. Composing invariants. In *Proc. FME*, volume 2805 of *LNCS*, pages 401–421. Springer, 2003.

[12] H. Foster, W. Emmerich, J. Kramer, J. Magee, D. S. Rosenblum, and S. Uchitel. Model checking service compositions under resource constraints. In *ESEC/SIGSOFT FSE*, pages 225–234. ACM, 2007.

[13] M. F. Frias, J. P. Galeotti, C. L. Pombo, and N. Aguirre. DynAlloy: Upgrading Alloy with actions. In *Proc. ICSE*. ACM, 2005.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[15] H. Giese. Modeling and verification of cooperative self-adaptive mechatronic systems. In *Reliable Systems on Unreliable Networked Platforms*, volume 4322 of *LNCS*. Springer Verlag, 2007.

[16] H. Giese, S. Henkler, M. Hirsch, M. Tichy, and H. Vöcking. Modellbasierte Entwicklung vernetzter, mechatronischer Systeme am Beispiel der Konvoifahrt autonom agierender Schienenfahrzeuge. In *Proc. of the 4th Paderborner Workshop Entwurf mechatronischer Systeme*, volume 189 of *HNI-Verlagsschriftenreihe*, 2006.

[17] H. Giese, M. Tichy, S. Burmester, W. Schäfer, and S. Flake. Towards the Compositional Verification of Real-Time UML Designs. In *Proc. ESEC/FSE*, pages 38–47. ACM, September 2003.

[18] R. Helm, I. M. Holland, and D. Gangopadhyay. Contracts: Specifying Behavioral Compositions in Object-Oriented Systems. In *Proc. OOPSLA/ECOOP '90*, pages 169–180, New York, NY, USA, 1990. ACM.

[19] D. Jackson. Alloy: A lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, 2002.

[20] H. J. Köhler, U. A. Nickel, J. Niere, and A. Zündorf. Integrating UML Diagrams for Production Control Systems. In *Proc. ICSE*, pages 241–251. ACM, 2000.

[21] F. Kordon. Mastering complexity in formal analysis of complex systems: Some issues and strategies applied to intelligent transport systems. *Proc. ISORC*, pages 420–427, 2007.

[22] J. Kramer and J. Magee. Self-Managed Systems: an Architectural Challenge. In *FOSE '07: 2007 Future of Software Engineering*, pages 259–268. IEEE Computer Society, 2007.

[23] P. Ölveczky and J. Meseguer. Specification and analysis of real-time systems using Real-Time Maude. In *Proc. FASE*, LNCS 2984, pages 354–358. Springer, 2004.

[24] F. Rammig. Engineering self-coordinating real-time systems. *Proc. ISORC*, pages 21–28, 2007.

[25] T. Reenskaug, P. Wold, and O. A. Lehne. *Working With Objects - The OOram Software Engineering Method*. Manning Publications Co., Greenwhich, CT 06830, UK, 1996.

[26] A. Rensink. Towards model checking graph grammars. In *Proc. AVoCS*, pages 150–160. University of Southampton, 2003.

[27] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation: Foundations*, volume 1. World Scientific Pub Co, 1997.

[28] R. T. Sanders, H. N. Castejon, F. A. Kraemer, and R. Braek. Using UML 2.0 collaborations for compositional service specification. In *Proc. MoDELS*, volume 3713 of *LNCS*, pages 460–475. Springer Berlin / Heidelberg, 2005.

[29] W. Schäfer and H. Wehrheim. The challenges of building advanced mechatronic systems. In *FOSE*, pages 72–84. IEEE Computer Society, 2007.

[30] D. Varró. Automated formal verification of visual modeling languages by model checking. *Software and System Modeling*, 3(2):85–113, 2004.

# Aktuelle Technische Berichte des Hasso-Plattner-Instituts

| Band | ISBN | Titel | Autoren / Redaktion |
|---|---|---|---|
| 28 | 978-3-940793-84-3 | **Efficient Model Synchronization of Large-Scale Models** | Holger Giese, Stephan Hildebrandt |
| 27 | 978-3-940793-81-2 | **Proceedings of the 3rd Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering** | Hrsg. von den Professoren des HPI |
| 26 | 978-3-940793-65-2 | **The Triconnected Abstraction of Process Models** | Artem Polyvyanyy, Sergey Smirnov, Mathias Weske |
| 25 | 978-3-940793-46-1 | **Space and Time Scalability of Duplicate Detection in Graph Data** | Melanie Herschel, Felix Naumann |
| 24 | 978-3-940793-45-4 | **Erster Deutscher IPv6 Gipfel** | Christoph Meinel, Harald Sack, Justus Bross |
| 23 | 978-3-940793-42-3 | **Proceedings of the 2nd. Ph.D. retreat of the HPI Research School on Service-oriented Systems Engineering** | Hrsg. von den Professoren des HPI |
| 22 | 978-3-940793-29-4 | **Reducing the Complexity of Large EPCs** | Artem Polyvyanyy, Sergy Smirnov, Mathias Weske |
| 21 | 978-3-940793-17-1 | **"Proceedings of the 2nd International Workshop on e-learning and Virtual and Remote Laboratories"** | Bernhard Rabe, Andreas Rasche |
| 20 | 978-3-940793-02-7 | **STG Decomposition: Avoiding Irreducible CSC Conflicts by Internal Communication** | Dominic Wist, Ralf Wollowski |
| 19 | 978-3-939469-95-7 | **A quantitative evaluation of the enhanced Topic-based Vector Space Model** | Artem Polyvyanyy, Dominik Kuropka |
| 18 | 978-3-939469-58-2 | **Proceedings of the Fall 2006 Workshop of the HPI Research School on Service-Oriented Systems Engineering** | Benjamin Hagedorn, Michael Schöbel, Matthias Uflacker, Flavius Copaciu, Nikola Milanovic |
| 17 | 3-939469-52-1 / 978-3-939469-52-0 | **Visualizing Movement Dynamics in Virtual Urban Environments** | Marc Nienhaus, Bruce Gooch, Jürgen Döllner |
| 16 | 3-939469-35-1 / 978-3-939469-35-3 | **Fundamentals of Service-Oriented Engineering** | Andreas Polze, Stefan Hüttenrauch, Uwe Kylau, Martin Grund, Tobias Queck, Anna Ploskonos, Torben Schreiter, Martin Breest, Sören Haubrock, Paul Bouché |
| 15 | 3-939469-34-3 / 978-3-939469-34-6 | **Concepts and Technology of SAP Web Application Server and Service Oriented Architecture Products** (noch nicht erschienen) | Bernhard Gröne, Peter Tabeling, Konrad Hübner |
| 14 | 3-939469-23-8 / 978-3-939469-23-0 | **Aspektorientierte Programmierung – Überblick über Techniken und Werkzeuge** | Janin Jeske, Bastian Brehmer, Falko Menge, Stefan Hüttenrauch, Christian Adam, Benjamin Schüler, Wolfgang Schult, Andreas Rasche, Andreas Polze |

ISBN 978-3-940793-91-1
ISSN 1613-5652