Cyber-Physical Systems with Dynamic Structure: Towards Modeling and Verification of Inductive Invariants

Basil Becker, Holger Giese

Technische Berichte Nr. 64

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 | 64

Basil Becker | Holger Giese

Cyber-Physical Systems with Dynamic Structure

Towards Modeling and Verification of Inductive Invariants

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.de/ abrufbar.

Universitätsverlag Potsdam 2012
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
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.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URL http://pub.ub.uni-potsdam.de/volltexte/2012/6243/
URN urn:nbn:de:kobv:517-opus-62437
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-62437

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-217-9

Abstract

Cyber-physical systems achieve sophisticated system behavior exploring the tight interconnection of physical coupling present in classical engineering systems and information technology based coupling. A particular challenging case are systems where these cyber-physical systems are formed ad hoc according to the specific local topology, the available networking capabilities, and the goals and constraints of the subsystems captured by the information processing part.

In this paper we present a formalism that permits to model the sketched class of cyber-physical systems. The ad hoc formation of tightly coupled subsystems of arbitrary size are specified using a UML-based graph transformation system approach. Differential equations are employed to define the resulting tightly coupled behavior. Together, both form hybrid graph transformation systems where the graph transformation rules define the discrete steps where the topology or modes may change, while the differential equations capture the continuous behavior in between such discrete changes. In addition, we demonstrate that automated analysis techniques known for timed graph transformation systems for inductive invariants can be extended to also cover the hybrid case for an expressive case of hybrid models where the formed tightly coupled subsystems are restricted to smaller local networks.

Contents

Chapter 1

Introduction

Cyber-physical systems (CPS) [10] exhibit sophisticated system behavior that results from the tight interconnection of physical coupling present in classical engineering systems and coupling via information technology that cannot be achieved by classical control or information technology alone.

A particular challenging case are systems where these cyber-physical systems are formed ad hoc according to the specific local topology, the available networking capabilities, and the goals and constraints of the subsystems captured by the information processing part.

In such cyber-physical systems with dynamic structure we have to cover two sources for complexity: At first we have cyber-physical subsystems tightly coupled locally via physical effects but also information technology. In addition, at the coarse-grain level we have arbitrary complex topologies that evolve over time and control which locally tightly coupled cyber-physical subsystems come into existence.

While the locally tightly coupled cyber-physical subsystems alone already result in complex hybrid systems [1] with finite discrete state space, the coarse-grain level results also in dynamic structures that evolve potentially infinitely and where the relevant initial configurations can be only characterized by some constraints but not collapsed into a finite set of initial configurations. Therefore, we in essence have to deal with hybrid systems that also have a infinite discrete state component.

1.1 Contribution

In this paper we present a formalism that permits to model the sketched class of cyber-physical systems. The ad hoc formation of tightly coupled cyber-physical subsystems of arbitrary size is specified using UML-based graph transformation systems. Modes, rules for mode changes, and differential equations defined for the modes are employed to define the tightly coupled cyber-physical behavior by means of hybrid behavior. Together, both options result in hybrid graph transformation systems where the graph transformation rules define the discrete steps such as formation of subsystems or mode changes, while the differential equations of the modes capture the continuous behavior in between such discrete changes.

Besides modeling the outlined class of cyber-physical systems, we of course also need means to predict their behavior and provide guarantees for crucial system properties. To this means, we demonstrate that automated analysis techniques for inductive invariants known for untimed and timed graph transformation systems can be extended to also cover the hybrid case for a subclass of the presented approach, where the tightly coupled cyber-physical subsystems result from only local reconfiguration rules and are of bounded size.

1.2 Former Work

The presented results extend former work that started with an approach to model and checking inductive invariants for graph transformation systems in [6]. This work was combined with result for the checking of timed coordination behavior in [13]. However, the structural rules and the real-time aspects had to provide the required guarantees independent of each other. Later, in [7] the modeling and checking concepts have been extended to timed graph transformation systems where real-time constraints for the ad hoc formation of coordination structures could be first modeled and also verified. In contrast to these earlier results, in the current paper the supported hybrid graph transformation systems permit that the complex hybrid behavior that results from the tight coupling via physical effects as well as information technology can be captured. In addition, also the checking procedures have been extended to a subclass of these hybrid systems.

1.3 Application Example

In this paper we will use the RailCab[1] system as running example to illustrate the presented approach. The RailCab system has been developed at the University of Paderborn and targets the development of a new railway technology (see Figure 1.1).

Figure 1.1: The test track and shuttle prototype of the RailCab project

The RailCab system's main constituents are so called Shuttles, which are small and autonomous vehicles and can either be used for cargo or persons. Customers place their transportation requests online and shuttle can reply to these requests by an offer. However, to be a competitive alternative to established public and individual transportation, the shuttles have to be fast and affordable. This can only be achieved if the shuttles consume less or at least not more energy per passenger than other means of transport. Unfortunately the energy consumption of a single shuttle, driving alone, is worse than that of classical trains. But it can be improved if multiple shuttles collaborate and build a convoy. Within the convoy only the first shuttle has to overcome the wind resistance and all following shuttles can benefit from the first shuttle's work. Thus in average the energy consumption of multiple shuttles is less compared to today's most energy efficient trains.

The convoy are not build by mechanical coupling but via information technology and thus a convoy is a nice example for a cyber-physical system where

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

besides the physical coupling effects also information technology plays a major role. In addition, the RailCab system requires that the convoys are established but also destructed according to the local situation and thus we have a cyber-physical system with dynamic structures.

Obviously the RailCab system is a safety critical system. As the shuttles are autonomous, the system's safety has to be guaranteed for this cyber-physical system with dynamic structures. In this paper we focus on the convoy building aspects of the RailCab system, demonstrate how it can be modeled with the presented approach, and prove that collisions between two following shuttles are excluded.

1.4 Outline

The report is structured as follows: In Section 2 we outline the modeling approach for hybrid graph transformation systems by means of the application example. Then, the automated verification approach permitting to verify inductive invariants is introduced and the verification of the application example is outlined in Section 3. The paper closes with a discussion of the related work in Section 4 and a final conclusion and outlook on future work.

Chapter 2

Modeling

For the modeling of the outlined class of cyber-physical systems with dynamic structure we extend graph transformation systems. These graph transformation systems are extended towards hybrid behavior by special nodes, which represent the laws of the system's continuos behavior. In a first step we will shortly introduce graph transformation systems and extend them in a second step to cover also the hybrid behavior.

2.1 Graph Transformation Systems

We employ UML class diagrams and story patterns for the modeling of the system's behavior. To allow the also the modeling of continuous behavior we will introduce in the next subsection special control modes, that represent different laws describing the continuous behavior.

2.1.1 Notation

A UML class diagram is used to model the system's type system and which associations are required and allowed to exist between the instances. In Figure 2.1 the class diagram showing the types for our application example is depicted. Shuttles are located at one Track per time and can be connected to other Shuttles through a DistanceCoordination pattern. The DistanceCoordination

pattern ensures a communication exchange, which is required to ensure safe driving in close proximity. Further, each Shuttle must have either a Speed- or PositionControlMode attached to it. These two control modes determine the way the Shuttle drives, by specifying the control laws for the Shuttle's attributes (details to this are presented in Section 2.2).

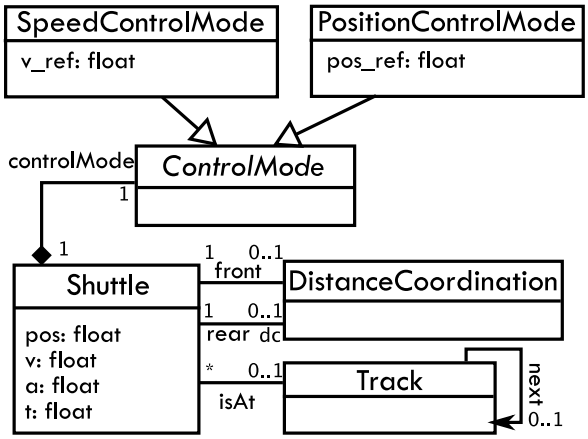

Figure 2.1: Class diagram of the RailCab system

For the modeling of the Shuttle's behavior we facilitate StoryDiagrams, a variant of UML collaboration diagrams, that are augmented with stereotypes to indicate side effects for creation and deletion of elements (cf. [17]). The stereotype ≪create≫ marks elements that will be created if the StoryDiagram is executed. The stereotype ≪delete≫ is defined analogously, but for the deletion of elements. All other elements – those that neither have a ≪create≫ nor a ≪delete≫ stereotype attached – are preserved by the StoryDiagram. The precondition of an StoryDiagram is given by the preserved elements and those that are to be deleted. Figure 2.2 depicts an example for a StoryDiagram specifying the movement of a Shuttle to the succeeding Track. This StoryDiagram deletes the association between Shuttle s1 and Track t1 and creates an association between Shuttle s1 and Track t2.

The StoryDiagram in Figure 2.2 contains only a constructive precondition, i.e. the precondition exactly states which elements have to exist. However, such preconditions are not expressive enough to express that, e.g., a Shuttle moves to an empty Track. To formulate StoryDiagrams like this it is required to use so called negative application conditions (NAC). NAC explicitly forbid the existence of the elements contained in the NAC. Figure 2.3 shows a StoryDiagram that specifies that a Shuttle is allowed is only allowed to move to the next Track if this Track does not have a Shuttle located on it.

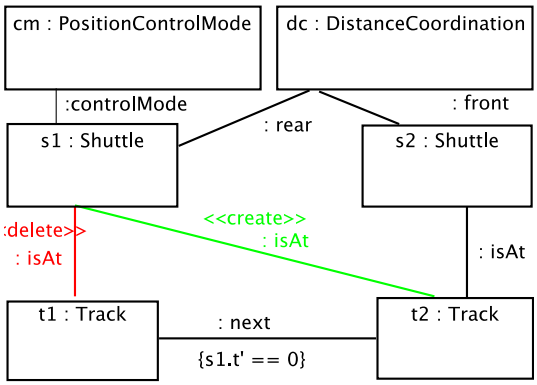

Figure 2.2: StoryDiagram moveDC

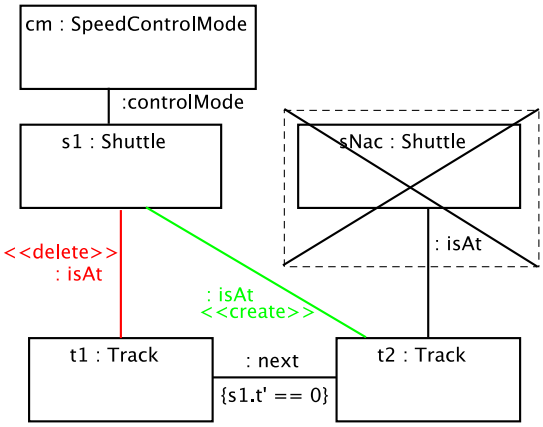

Figure 2.3: StoryDiagram for the move rule including a NAC

We use StoryDiagrams to specify the unsafe states of the system, too. However, these StoryDiagrams must not contain side effects. In our application example we want to forbid states where two Shuttles are located at the same Track without having a DistanceCoordination pattern instantiated between them (cf. Figure 2.4).

2.1.2 Formal Model

After having introduced the concrete syntax of our modeling language we will use the following paragraphs to specify the formal semantics of StoryPattern and forbidden states.

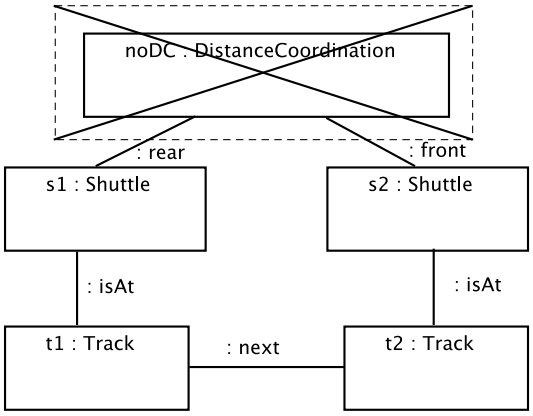

Figure 2.4: Forbidden pattern noDC

Graph transformation systems (GTS) are a well established way to specify behavior of a system, whose state can be expressed as a graph. This holds for object-oriented and component based systems as well as for networks. A graph G is formally specified as $G = (V, E, l_v, l_e)$ where V is a finite set of vertices, $E \subseteq V \times V$ is a set of edges and l_v, l_e are labeling functions for the nodes and edges, respectively. The labeling functions l_v and l_e assign each node and each edge a label from the global alphabet $\mathcal{A}$. In our application example the alphabet $\mathcal{A}$ is given as

$$\mathcal{A} = \{Track, Shuttle, DistanceCoordination, PositionControlMode, \\ SpeedControlMode, isAt, next, front, rear, controlMode\}.$$

The set of all graphs is denoted $\mathcal{G}$. A graph isomorphism is a bijective mapping function between two graphs, which preserves the graphs' type and structural constraints. Let $G, H \in \mathcal{G}$ be Graphs and $m = (m_V, m_E)$ a mapping from G to H. This mapping is a graph isomorphism if and only if

$$\begin{aligned} \forall v \in V_G : \exists v' : v' \in V_H \wedge (v, v') \in m_v \wedge \\ l_v(v) = l_v(v') \\ \forall e \exists e' : e \in E_G \rightarrow e' \in E_H \wedge \\ (e, e') \in m_e \wedge l_e(e) = l_e(e') \\ \forall (e = (s, t), e' = (s', t')) : (e, e') \in m_e \rightarrow \\ (s, s') \in m_v \wedge (t, t') \in m_v \end{aligned}$$

We write $G \approx_m H$ if the isomorphism m maps the graph G to the graph H.

A graph pattern represents a possibly infinite number of graphs. A graph pattern is formally specified as $P = (P^+, P^-)$ with $P^+ \in \mathcal{G}$ and $P^- =\in \mathcal{G}$. A

graph pattern $P = (P^+, P^-)$ can be matched to a graph $G \in \mathcal{G}$ if a subgraph of G' of G exists, such that $G' \approx_m P^+$ and the subgraph G' could not extended in such a way, that any of the elements contained in P^- could be matched, too. A graph pattern P matches a graph pattern Q if there exist two isomorphic functions m^+ and m^- that map all elements from P^+ to the elements of Q^+ and all elements from P^- to Q^-.

A graph transformation $P = (L, R, i)$ can be specified through two graph patterns $L = (L^+, L^-)$ and $R = (R^+, \emptyset)$ and a graph isomorphism $i : L^+ \mapsto R^+$. The graph isomorphism i identifies those elements in L^+ and R^+ that are preserved by the graph transformation. Following, the elements, that are deleted by the graph transformation P are given as $del_P = L^+ \backslash dom(i)$ and the created elements are given as $new_P = R^+ \backslash ran(i)$. A graph transformation can be applied to a graph H if we can find an isomorphism a that maps the graph rule's left hand side $L = (L^+, L^-)$ to H. The result of the application is specified as $H' = H \backslash (del_P \circ a) \cup (new_P \circ a)$. We write graph rule applications as $H \xrightarrow{P,a} H'$. We can define a GTS $S = (\mathcal{R}, prio)$ with $\mathcal{R}$ being a set of graph transformations and $prio : \mathcal{R} \mapsto \mathbb{N}$ is a priority function that assigns each rule a priority. A transition $G \xrightarrow{r}_S H$ under the restrictions of the GTS S exists iff $G \xrightarrow{r} H$ is a valid application of r and $\nexists p : p \in \mathcal{R} \wedge prio(p) > prio(r) \wedge G \xrightarrow{p} H'$. We say H is reachable from G. A sequence of multiple rule applications under the restrictions of S is written as $G \rightarrow_S^* H$ and following we can define REACH$(S, G_0) = \{H | G_0 \rightarrow_S^* H\}$.

2.2 Hybrid Graph Transformation systems

2.2.1 Notation

The notation of graphs, graph transformation and forbidden patterns is mostly in conformity with the notation we have presented in Section 2.1. However, we add constraints to the graph transformations and forbidden patterns to restrict their applicability. These conditions have to be linear, thus allowing only products of variables and real valued coefficients. In Figure 2.2 the rule moveDC is restricted to situations where the shuttle's position has reached the current track's end. After the rule has been applied the shuttle's attribute t – a clock measuring the time the shuttle has already spent at the current track – is reset to zero.

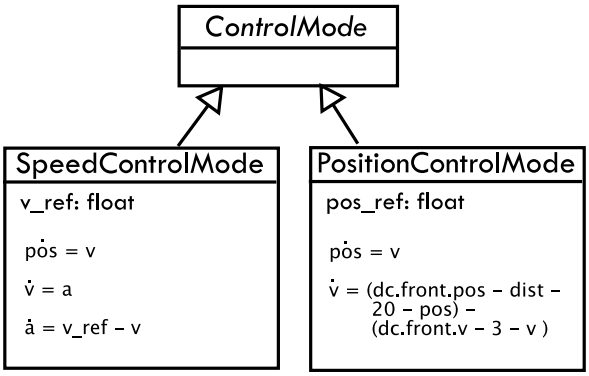

Figure 2.5: Clas diagram snippet showing the control laws

2.2.2 Formal Model

To capture also continuous behavior we introduce attributes and control laws that constrain the attribute's continuous development. The global set Var contains all available attributes. Each attribute is assigned to a type from the global alphabet $\mathcal{A}$. This assignment is specified in $Attr : \mathcal{A} \mapsto 2^{Var}$ with $a, a' \in \mathcal{A}\, Attr(a) \cap Attr(a') = \emptyset$ for $a \neq a'$.

Obviously, graphs as defined above are not well suited to represent the state of a hybrid system. Thus, we extend graphs to also hold information of the system attributes' values. A hybrid graph is given as $G = (V, E, l_v, l_e, X, \beta, \theta)$. The constituents V, E, l_v, l_e are defined as known. The set $X \subseteq V \times Var$ with $\forall (v, i) : (v, i) \in X \rightarrow i \in Attr(l_v(v))$ contains all variables existing in the graph G. The function $\beta : X \mapsto \mathbb{R}$ assigns all occurring variables a real value. The constraint $\theta : X \cup \dot{X} \mapsto \mathbb{B}$ with $\dot{X}$ representing the first derivative with respect to the time of the variables contained in X.

Let $G = (V, E, l_v, l_e, X, \beta, \theta)$ be a hybrid graph and $CM \subseteq V$ the set of control nodes contained in G. Each control mode $v_{CM} \in CM$ specifies a constraint $\theta_{v_{CM}}$ over the variables $X_{v_{CM}} \subseteq X_G$ and $\dot{X}_{v_{CM}}$. The nodes holding these variables must be reachable from v_{CM}. We can now define $\theta_G = \bigwedge_{v_{CM} \in CM} \theta_{v_{CM}}$. In our application example we have two possible control modes (cf. Figure 2.5) that can only be connected to Shuttles: SpeedControlMode and PositionControlMode. If the Shuttle is in SpeedControlMode $\theta_{v_{VM}}$ is given as $\dot{pos} = v \wedge \dot{v} = a \wedge \dot{a} = v_{ref} - v$.

A hybrid graph transformation is given as $P = (L, R, i, \phi)$ with $\phi : X_{L^+} \cup X_{R^+} \mapsto \mathbb{B}$ being a linear constraint, that restricts the transformation's appli-

cability. The transformation $H \xrightarrow{P,a} H'$ from the hybrid graph H to the hybrid graph H' is correct if H, H', P and a meet all requirements for graph transformations as defined in Section 2.1 and $\phi(\beta_{H|dom(a)}, \beta_{H'|ran(a)}) \equiv true$ with $\beta_{H|dom(a)}$ being H's valuation restricted to the isomorphism's domain, and $\beta_{H'|dom(a)}$ being H''s valuation restricted to the isomorphism's range. If want to explicitly mention that H and H' are hybrid graphs we can also write $(H, \beta) \xrightarrow{P,a} (H', \beta')$.

A hybrid graph pattern $P = (P^+, P^-, \phi)$, where ϕ is a constraint over the variables X_{P^+}, matches a hybrid graph H under the isomorphism m if there non hybrid counterparts match as described above and additionally the graph's valuation β fulfills the pattern's condition ϕ. A hybrid graph pattern $P = (P^+, P^-, \phi_P)$ matches a hybrid graph pattern $Q = (Q^+, Q^-, \phi_Q)$ if we find isomorphisms between P and Q and $\forall \beta : \beta \in X_{Q^+} \times V_{Q^+} \mapsto \mathbb{R} \wedge \phi_Q(\beta) \implies \phi_P(\beta_m)$. With $\beta_m = \{(v, r) | (v', r) \in \beta \wedge (v, v') \in m_v\}$ being the translation of β according to the isomorphism $m = (m_v, m_e)$. We write $P \sqsubseteq Q$ or again $(P, \phi_P) \sqsubseteq (Q, \phi_Q)$ to explicitly stress the matching of hybrid graph pattern.

Given a set of hybrid graph transformations we can combine them to a hybrid graph transformation system (HGTS) $S = (\mathcal{R}, prio, \mathcal{R}_u)$ with $\mathcal{R}$ being a set of hybrid graph transformations, $prio : \mathcal{R} \mapsto \mathbb{N}$ a function that assigns each rule a priority (rules can be preempted by other rules having a higher priority) and $\mathcal{R}_u \subseteq \mathcal{R}$ is a set of urgent rules. If a rule is marked as urgent the rule has to be applied as soon as the rule is applicable. In a HGTS $S = (\mathcal{G}, prio, \mathcal{R}_u)$ a discrete transition $(G, \beta_G) \xrightarrow{R} (H, \beta_H)$ is allowed if no rule $P \in \mathcal{R}$ with $prio(P) > prio(R)$ and $(G, \beta_G) \xrightarrow{P} (H', \beta_{H'})$ exists. A continuous transition$(G, \beta_G) \xrightarrow{\delta} (G, \beta'_G)$ with duration $0 \leq \delta$ is allowed if and only if a function $f : \mathbb{R} \mapsto (X_G \mapsto \mathbb{R})$ exists that is differentiable in the closed interval $[0 \ldots \delta]$, with $f(0) = \beta_G$ and $f(\delta) = \beta'_G$, such that $\theta(f(t), \dot{f}(t)) \equiv true$ for every $0 \leq t \leq \delta$ and not exists t' with $0 \leq t' < \delta$ such that $(G, f(t')) \xrightarrow{R}$ for any $R \in \mathcal{R}_u$. We use $\phi \ominus x$ for $x \leq \delta$ to denote the continuous change of the attributes over time for the time difference x according to the differential equations starting with the constraint assignment space ϕ ($\phi \ominus x = \beta'_G | (G, \beta_G) \xrightarrow{x} (G, \beta'_G) \wedge \beta_G \models \phi$ for the given G). We write $G \rightarrow_S H$ to denote that the HGTS specifies a transition (either continuous or discrete) from G to H. If the graph H can be reached from G by a sequence of transitions we write $G \rightarrow_S^* H$. Finally the set REACH$(S, G_0) = \{G | G_0 \rightarrow_S^* G\}$ defines the reachable graphs, given the start graph G_0 and the HGTS S.

Chapter 3

Verification of Inductive Invariants

The UML-based modeling introduced in Section 2 and the underlying formal HGTS allow us to extend an existing verification technique for GTS which covers inductive invariants for the possible structural changes. Our former approach [6] for GTS is first explained and we then outline its extension towards HGTS. Furthermore, results for the checking of the RailCab example using the new technique are presented.

3.1 GTS Case

A set of forbidden graph patterns $\mathcal{F} = \{F_1, \ldots, F_n\}$ are employed in the underlying approach [6] for GTS to represent those cases of the system that have to be excluded (hazards, accidents, incidents). We say that the property $\Phi_{\mathcal{F}}$, denoted by $G \models \Phi_{\mathcal{F}}$, holds iff G matches none of the forbidden graph patterns in $\mathcal{F}$. If a graph G matches a forbidden graph pattern $F \in \mathcal{F}$, we call G a *witness* for the inverted property $\neg\Phi_{\mathcal{F}}$.

$\Phi_{\mathcal{F}}$ is an *operational invariant* for the GTS S iff for all $G \in \mathsf{REACH}(S, G^0)$ for a given initial graph G^0 holds $G \models \Phi_{\mathcal{F}}$ (cf. [8]). As graph transformation systems with types are Turing-complete, checking them is restricted to finite models. As the considered systems fall not into this category, we instead tackle the problem whether $\Phi_{\mathcal{F}}$ is only an *inductive invariant* which is the case if for all graphs G and for all rules $r \in \mathcal{R}$ holds that $G \models \Phi_{\mathcal{F}} \ \wedge \ G \xrightarrow{r} G'$ implies

$G' \models \Phi_{\mathcal{F}}$. It is to be noted that an inductive invariant implies the related operational invariant but not vice versa as inductive invariants are stronger.

In our case we can reformulate the conditions for an *inductive invariant* to have a falsifiable form as follows: $\Phi_{\mathcal{F}}$ is an inductive invariant of a GTS $S = (\mathcal{R}, prio)$ iff there exists no pair (G, r) of a graph G and a rule $r \in \mathcal{R}$ with $G \models \Phi_{\mathcal{F}}$, $G \xrightarrow{r} G'$ and $G' \not\models \Phi_{\mathcal{F}}$. If in contrast a pair (G, r) witnesses the violation of property $\Phi_{\mathcal{F}}$ by rule r, we have a *counterexample* for $\Phi_{\mathcal{F}}$.

The application of a rule can only have a local effect (cf. [6]). We exploit this fact to verify whether a counterexample (G, r) exists. It can only exist when the rule is not preempted by one with a higher priority and the local modification of G by rule r is transforming the correct graph G into a graph that violates the property. By representing the potentially infinite many possible counterexamples by an only finite set of representative set $\Theta(R_l, F_i)$ of graph patterns P' (each a combinations of a RHS R_l of a rule r_l and a forbidden graph pattern $F_i \in \mathcal{F}$; cf. [6]), we can only consider a finite number of cases to check that no counterexample exists (and $\Phi_{\mathcal{F}}$ is thus an inductive invariant).

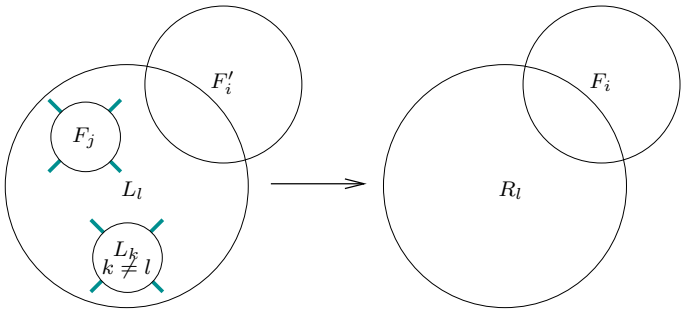

Figure 3.1: Schema to check a potential counterexample (P, r_l) with resulting graph pattern P' that is a combination of a RHS R_l of a rule r_l and a forbidden graph pattern $F_i \in \mathcal{F}$ (cf. [7])

As depicted in Figure 3.1 to do so, we have to check for some $F_i \in \mathcal{F}$ and $r_l \in \mathcal{R}$ for any graph pattern $P' \in \Theta(F_i, R_l)$ whether the pair (P, r_l) with P defined by $P \xrightarrow{r} P'$ is a counterexample for $\Phi_{\mathcal{F}}$ or not as follows:

1. Check first that the rule r_l can be applied to graph pattern P at all and that the resulting graph pattern is P'. this requires that no other rule $r_k \in \mathcal{R} \setminus \{r_l\}$ with higher priority ($prio(r_k) > prio(r_l)$) exists that matches P.

2. Check in addition that there exists no $F_j \in \mathcal{F}$ with $F_j \sqsubseteq P$ as otherwise P is already invalid.

The checking algorithm has to perform this check for any given rule $(L, R)_r \in \mathcal{R}$ and forbidden graph pattern $F \in \mathcal{F}$. It first computes the set of all possible target graph patterns for R and the forbidden graph pattern F ($\Theta(R, F)$) and then computes the related source graph patterns. The conditions are then checked for all source graph pattern to determine if the pair (P, r) is a valid counterexample. In [6] an explicit as well as symbolic algorithm for GTS along these lines have been presented.

In Figure 3.4 a pair of source graph pattern and target graph pattern is shown. The pair has been created by combining the right hand side of the moveDC rule (cf. Figure 2.2) and the forbidden subgraph collision.

3.2 HGTS Case

To extend the checking scheme to the extension of the modeling technique for continuous and hybrid behavior outlined in Section 2, we have to take into account that the behavior is described by a combination of rule applications and time steps where the continuous dynamic evolves. Consequently, reaching a forbidden graph pattern could involve a rule application as well as a time step. The trick to approach the checking is to extend the untimed case similar to [7]. At first we have to determine for which graph pattern the forbidden graph pattern might be reached. Then, we habe to check whether for this case the combination between the rule application and time steps could really lead from a valid configuration to an invalid one using a hybrid automata model checker.

We can analogously to the untimed case formulate the definition of an *inductive invariant* for the hybrid case in a falsifiable form: $\Phi_{\mathcal{F}}$ with forbidden hybrid graph pattern $(F_i, \psi_i) \in \mathcal{F}$ is an inductive invariant of a HGTS $S = (\mathcal{R}, \mathcal{R}_u, prio)$ iff no pair $((G, \alpha), r)$ of an hybrid graph (G, α) and an hybrid rule $r \in \mathcal{R}$ an a time length δ exists such that $(G, \alpha) \models \Phi_{\mathcal{F}}$, $(G, \alpha) \xrightarrow{r}\xrightarrow{\delta} (G', \beta)$, and $(G', \beta) \not\models \Phi_{\mathcal{F}}$. Such a pair $((G, \alpha), r)$ which witnesses the violation of property $\Phi_{\mathcal{F}}$ by rule r is then a *counterexample* for the hybrid case.

Using the same idea as for the untimed case we can lift this problem to hybrid graph pattern. Again, only a finite set of representative hybrid 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 $r_l = ((L_l, \phi_l), R_l, \mu_l)_{r_l}$ and a forbidden graph pattern $(F_i, \psi_i) \in \mathcal{F}$ have to be considered.

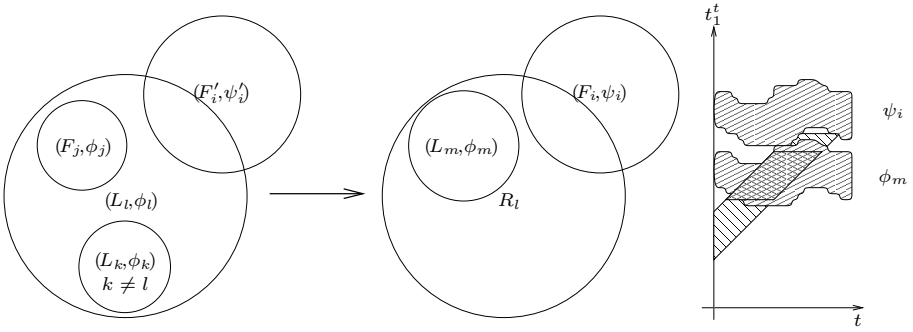

Figure 3.2: Schema to check a potential counterexample $((P, \phi_P), r_l)$ with resulting graph pattern $(P', \phi_{P'})$ that is a combination of a RHS R_l of a rule r_l and a forbidden graph pattern $(F_i, \psi_i) \in \mathcal{F}$ in the hybrid case (cf. [7])

For the hybrid case we have to check for any hybrid 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 $r_l \in \mathcal{R}$ as depicted in Figure 3.2 whether the pair $((P, \phi_P), r_l)$ with (P, ϕ_P) defined by $(P, \phi_P) \xrightarrow{r} \xrightarrow{\delta} (P', \phi_{P'})$ is a counterexample for $\Phi_{\mathcal{F}}$ as follows:

1. Check first that the rule r_l can be applied to hybrid graph pattern (P, ϕ_P) and that the $(P', \phi_{P'})$ results from this application plus a time step of length $\delta \geq 0$. Note that this requires that no $r_k \in \mathcal{R}_u \setminus \{r_l\}$ exists with $prio(r_k) > prio(r_l)$ that matches (P, ϕ_P) and that $(P', \phi_{P'} \ominus x)$ can really be reached as for all $x \leq \delta$ holds that $(P', \phi_{P'} \ominus x)$ is matched by no $r_m \in \mathcal{R}_u$.

2. Check secondly that there exists no $(F_j, \phi_j) \in \mathcal{F}$ with $(F_j, \phi_j) \sqsubseteq (P, \phi_P)$ as otherwise (P, ϕ_P) is already invalid.

The extended checking algorithm employs in its first step a slightly adjusted version of the untimed algorithm to derive potential counterexamples (see Figure 3.2). For the potential counterexamples a hybrid model checker is used to encode whether it is a real counterexample.

In the untimed case, it was sufficient to check whether the target graph pattern can be reached to judge whether the forbidden graph can be reached. In the hybrid case urgent rules may in fact prevent that we reach a state which

fulfills the conditions of the embedded forbidden hybrid graph patterns. Therefore, the encoding in form of a hybrid automata include transitions to a special urgent state that capture the behavior of urgent rules with higher priority ((L_k, ϕ_k)). Therefore, in the hybrid automata model a path to the forbidden hybrid pattern can only occur when no such higher priority rule has to be executed before.

Also the initial configuration before the rule application has to exclude that any forbidden hybrid graph pattern ((F_j, ψ_j)) is already present as otherwise we would not have the required transition from a valid configuration to an invalid one. This can be encoded by initial constraints for the initial state of the constructed hybrid automata.

Due to both steps the hybrid automata can then be checked by the model checker to prove whether a rule application and subsequent time step could really result in an invalid configuration starting from a valid one (whether we have a real counterexample).

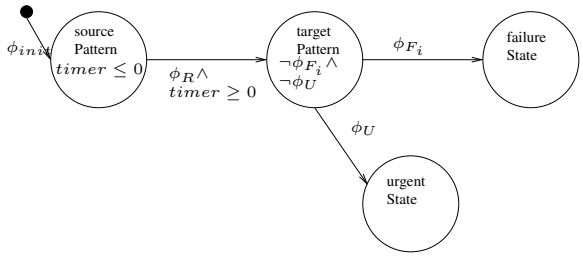

Figure 3.3: Generic automata for verifying the system's continuous part

In Figure 3.3 the generic translation scheme to hybrid automata for some rule (L_R, R_R, ϕ_R) and some forbidden pattern (F_i, ϕ_{F_i}) is depicted. The initial condition ϕ_{init} guarantees that the source graph pattern is safe, ϕ_U holds if one urgent transition is active. Let $\mathcal{F}_S \subseteq \mathcal{F}$ be the set of forbidden pattern that could be mapped into the source graph pattern and $U \subseteq \mathcal{R}_u$ the set of urgent transitions, that could be mapped to the target graph pattern. Then we can give ϕ_{init} as $\phi_{init} = \phi_R \wedge \bigwedge_{F_s \in \mathcal{F}_S} \neg\phi_{F_s}$ and ϕ_U as $\phi_u = \bigvee_{R_u \in U} \phi_{R_u}$. Thus the failureState location can only be reached if we start in a correct source graph pattern, apply the rule and wait until the condition ϕ_{F_i} holds. We use a clock called timer to force the hybrid automata to immediately leave the location initialState.

3.2.1 Verification Results

The verification of the rules' structural parts yields multiple possible witnesses against the system's correctness. As mentioned above our verification approach is twofold and the structural analysis gives us characteristic scenarios that could result in an unsafe situations. However, to be sure that the found example prove the system to be incorrect we have to show that starting in a safe source graph pattern the application of the rule brings the system into a situation where the condition of the forbidden pattern is reachable. With respect to possibly activated rules having a higher priority and urgent rules.

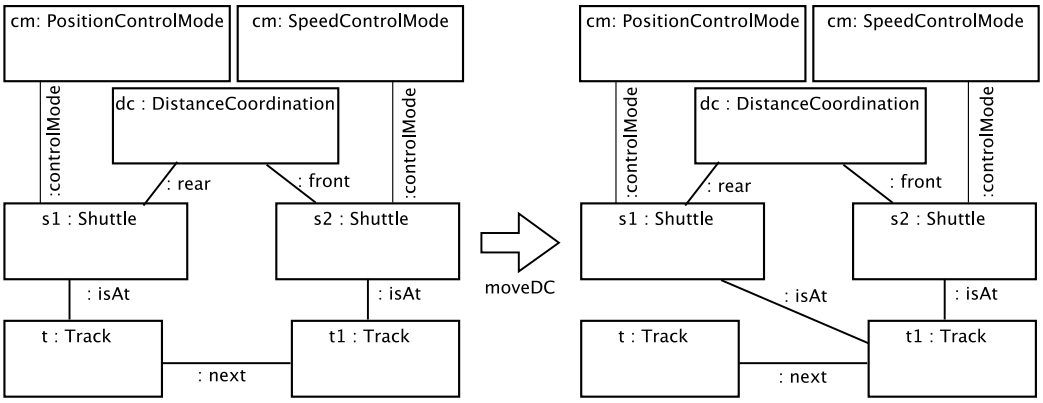

Figure 3.4: Possible counter example, derived from rule moveDC and forbidden pattern collision

One possible counter example the structural verification found is depicted in Figure 3.4. The figures left hand side shows the source graph pattern and the right hand side the target graph pattern, respectively. The target graph pattern has been created by overlapping the right hand side of rule moveDC and the forbidden pattern collision at the nodes s1, s2 and t2, thus the whole pattern collision could be found in the rule's right hand side. Reverse application of the rule led to the source graph pattern. In the source graph pattern we can find only one forbidden pattern: The collision of two Shuttles while on two succeeding Tracks.[1] We require this forbidden pattern as the Shuttle's length is supposed to be positive and thus it can happen that a Shuttle is physically at two Tracks at the same time. In our system the Shuttle is only allowed to be at one Track at one point in time (we use the Shuttle's front to determine the current track).

[1] Note that the forbidden pattern noDC (cf. 2.4) does not match due to the existence of the DistanceCoordination instance.

Listing 3.1: Hybrid automaton to counter example from Figure 3.4

```
automaton GenericHybridGTS
  contr_var: s1_pos, s2_pos, s1_v, s2_v, s1_a, s2_a, v_ref, pos_ref,
     timer;
  parameter: distance, failure;
  synclabs: void;
  loc sourcePattern: while t<=0 wait {timer' == 1};
    when timer >= 0 sync void do {pos_ref' == s1_pos -
       distance - 2 & s1_pos' == s1_pos & s2_pos' == s2_pos &
       s2_v' == s2_v & s1_v' == s1_v & s1_a' == s1_a & s2_a'
       == s2_a & v_ref' == v_ref & failure' == 0} goto
       targetPattern;
  loc targetPattern: while s1_pos - distance - s2_pos >= 0
     wait {s1_pos' == s1_v & s1_v' == s1_a & s1_a' == P * (
     v_ref - s1_v) & s2_v' == P_2 * (s1_pos - distance - 10 -
     s2_pos) - Q_2 * (s2_v - 3 - s1_v) & pos_ref' == s1_pos' &
      v_ref' == 0};
    when s1_pos - distance - s2_pos <= 0 sync void do {failure
       ' == 1} goto failureState;
  loc failureState: while true wait {true};
  loc urgentTransition: while true wait {true};
  initially: sourcePattern & s1_pos > s2_pos + distance + 10 &
      s2_pos > 0 & 60 < v_ref & v_ref < 200 & 60 < s1_v & s1_v
     < 200 & 3 <= s1_v - s2_v & s1_v - s2_v <= 3 & failure ==
     0 & 5 < distance & distance < 10;
end
```

In the target graph pattern we do not find an urgent rule that could restrict the reachability of the failure state. Thus, the hybrid automaton we have to check using PHAVer is the one shown in Listing 3.1. In this automaton mainly three locations are of interest: sourcePattern, targetPattern and failureState. The automaton's initial state is given by the line starting with **initially** :... . In the initial state all velocities are in their boundaries (we assume that velocities are less than 200 km/h) and that the shuttles s1 and s2 have not collided, yet. The situation we want to verify starts with the moment in time, when the rule is applied. Hence, the automaton has to leave the location sourcePattern immediately. We can express this in PHAVer with the use of a timer variable and a corresponding location invariant. Initially the timer is set to zero and the invariant is given as $timer \leq 0$. The guards of the transitions leaving the sourcePattern location are fulfilled if $timer \geq 0$ holds.

Following the active transition to the targetPattern location only resets the shuttle's timer t, which stores the time the shuttle is at the current Track. All other statements within the transitions **do** condition specify that no other attribute values are changed. The automaton can stay at the targetPattern location as long as the forbidden pattern's condition is not fulfilled, i.e. the Shuttles have not collided. If the forbidden pattern's property is fulfilled the transition to the failureState location becomes activated. This transition changes the value of the automaton's failure parameter to 1. The reachability analysis we perform using PHAVer checks, whether a state is reachable where the failure parameter's value is set to 1. Performing this analysis on the automaton shown in Listing 3.1 we find out that such a state (with failure set to 1) is not reachable, within the automaton. Thus, we showed that the possible counter example, provided by the structural analysis, is not a witness against the system's correctness.

3.3 Complete Checking

The complete algorithm performs this check for any given rule $((L_l, \phi_l), R_l, \mu_l)_{r_l} \in \mathcal{R}$ and forbidden graph pattern $(F_i, \psi_i) \in \mathcal{F}$ by computing the related set of all possible target graph patterns ($\Theta((F_i, \psi_i), R_l, \mu_l)$ and then derives the related source graph patterns. The above outlined cases are then employed to decide whether the source graph pattern (P, ϕ_P) represents potentially safe graphs that can be transformed into unsafe graphs by applying r plus a time step δ. If so, the pair $((P, \phi_P), r)$ is a valid counterexample.

In the application example we had a total of six rules and fourteen forbidden patterns. We had to introduce several forbidden pattern to show that the multiplicity constraints implicitly introduced through the system's class diagram are satisfied, too. We further modeled rules that allow the Shuttles to accelerate and decelerate if they are in SpeedControlMode. However, the algorithm had to check 64 pairs of rules and forbidden pattern. While he did this, he found 37 possible counterexamples, which we manually translated into a hybrid automaton for PHAVer and checked whether the failureState location could be reached. Between the possible counter examples were lots of similarities, which were mainly introduced due to isomorphism.

Chapter 4

Related Work

A number of related approaches for the verification of systems with structural changes like our earlier work [6] exist which do not support time dependent behavior. Further some approaches directly adresses the verification of hybrid systems, mostly relying on hybrid automata for the input specification. DynAlloy [12] extends Alloy [16] 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. An approach which has been successfully applied to verify service-oriented systems [4] is the one of Varró et al. It transforms visual models based on graph theory into a model-checker specific input [20]. A more direct approach is GROOVE [19] by Rensink where the checking works directly with the graphs and graph transformations. However, 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 abstraction to a finite state model of moderate size exist).

There are only first attempts that address the verification of infinite state systems with changing structure: In [3] 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 [5] 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. It is to be noted that in addition to the mentioned limitations, both approaches do not support time as approached in this paper.

The only approach we are aware of that addresses structural changes as well as time is Real-Time Maude [18] which is based on rewriting logics. 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. Again, the requirement of having an initial configuration and the limitation to finite state models excludes that the real-time coordination of autonomous vehicles can be fully covered.

Concerning the direct verification of hybrid system the work of Henzinger et al. [2, 15] is the first to mention. However, although we could have in principle used the tool HyTech we decided to use the improved implementation by Frehse [11] called PHAVer. Both tools are applicable for the verification of the counterexamples the structural analysis presents but are not applicable to the verification task in general.

There exist many approaches to cover the modeling of complex hybrid behavior sucvh as CHARON, Masaccio, HybridUML, UMLh, HyROOM, HyCharts, Mechatronic UML or Ptolemy (cf. [14]). However, none of them provides the capability to describe dynamic structures as required for the considered class of cyber-physical systems.

Chapter 5

Conclusion and Future Work

We have presented an approach that is able to model cyber-physical systems with dynamic structure. By extending established graph transformation systems theory towards hybrid systems, the approach permits to capture such systems at a reasonable high level of abstraction and to describe how the ad hoc formation of tightly coupled cyber-physical subsystems can happen.

Furthermore, the also presented extended checking approach for inductive invariants enables us to provide guarantees for such cyber-physical systems with dynamic structure if the number of different types (not instances) of tightly coupled cyber-physical subsystems is not too large.

5.1 Future Work

Future work will look for more convenient means to specify the local hybrid behavior in form of models and differential equations (e.g., supporting algebraic ones) as well as exploiting more sophisticated analysis tools for the checks required for the inductive invariants that would allow us to relax the constraints on the differential equations for checking inductive invariants.

For the case of discrete GTS we have recently developed an algorithm to check the existence of patterns, i.e. application of a set of graph-rules preservers the patterns, [9], in contrast to showing the absence of forbidden pat-

terns, as we have done in this report. We still have to evaluate to which extent this approach has to be adopted to be applicable for HGTS, too.

We also plan to apply our approach to a cyber physical system of small robots to evaluate the applicability of our approach. An important aspect of this is to ensure that the verified system models remain valid models of the running code. One possibility to narrow the gap between model and code is the implementation of an story diagram interpreter for embedded systems.

Bibliography

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

[2] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic Symbolic Verification of Embedded Systems. *IEEE Transactions on Software Engineering*, 22:181–201, 1996.

[3] Paolo Baldan, Andrea Corradini, and Barbara König. A Static Analysis Technique for Graph Transformation Systems. In *Proc. CONCUR*, volume 2154 of *LNCS*, pages 381–395. Springer, 2001.

[4] Luciano Baresi, Reiko Heckel, Sebastian Thöne, and Daniel Varró. Modeling and Validation of Service-Oriented Architectures: Application vs. Style. In *ESEC/FSE-11: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 68–77, New York, NY, USA, 2003. ACM.

[5] Jörg Bauer and Reinhard Wilhelm. Static Analysis of Dynamic Communication Systems by Partner Abstraction. In *Proceedings of the 14th International Symposium, SAS 2007, Kongens Lyngby, Denmark, August 22-24, 2007*, volume 4634 of *Lecture Notes in Computer Science*, pages 249–264. Springer Berlin / Heidelberg, 2007.

[6] Basil Becker, Dirk Beyer, Holger Giese, Florian Klein, and Daniela Schilling. Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation. In *Proc. of the* 28^th *International Conference on Software Engineering (ICSE), Shanghai, China*. ACM Press, 2006.

[7] Basil Becker and Holger Giese. On Safe Service-Oriented Real-Time Coordination for Autonomous Vehicles. In *In Proc. of 11th International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC)*, pages 203–210. IEEE Computer Society Press, 5-7 May 2008.

[8] Michel Charpentier. Composing Invariants. In *Proc. of International Symposium of Formal Methods Europe*, volume 2805 of *Lecture Notes in Computer Science*, pages 401–421. Springer, 2003.

[9] Johannes Dyck. Increasing expressive power of graph rules and conditions and automatic verification with inductive invariants. Master's thesis, Hasso-Plattner-Institut für Softwaresystemtechnik, Universität Potsdam, July 2012.

[10] National Science Foundation. Program Announcements & Information: Cyber-Physical Systems, September 2008. `http://www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf08611`.

[11] Goran Frehse. PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. In *Hybrid Systems: Computation and Control*, volume 3414 of *Lecture Notes in Computer Science*, pages 258–273. Springer Berlin / Heidelberg, 2005.

[12] Marcelo Fabian Frias, Juan Pablo Galeotti, Carlos Lopez Pombo, and Nazareno Aguirre. DynAlloy: Upgrading Alloy with actions. In *Proc. of International Conference of Software Engineering*, pages 442–451. ACM, 2005.

[13] Holger Giese. Modeling and Verification of Cooperative Self-adaptive Mechatronic Systems. In Fabrice Kordon and Janos Sztipanovits, editors, *Reliable Systems on Unreliable Networked Platforms - 12th Monterey Workshop 2005 . Laguna Beach, CA, USA, September 22-24,2005 . Revised Selected Papers*, volume 4322 of *Lecture Notes in Computer Science*, pages 258–280. Springer Verlag, 2007.

[14] Holger Giese and Stefan Henkler. A Survey of Approaches for the Visual Model-Driven Development of Next Generation Software-Intensive Systems. *Journal of Visual Languages and Computing*, 17(6):528–550, December 2006.

[15] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HyTech: A Model Checker for Hybrid Systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[16] Daniel Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, 11(2):256–290, 2002.

[17] Hans J. Köhler, Ulrich A. Nickel, Jörg Niere, and Albert Zündorf. Integrating UML Diagrams for Production Control Systems. In *Proc. of the* $22^n d$ *International Conference on Software Engineering (ICSE), Limerick, Ireland*, pages 241–251. ACM Press, 2000.

[18] Peter Ölveczky and José Meseguer. Specification and Analysis of Real-Time Systems Using Real-time Maude. In Tiziana Margaria and Michel Wermelinger, editors, *Proceedings on Fundamental Approaches to Software Engineering (FASE2004)*, volume 2984 of *Lecture Notes in Computer Science*. Spinger-Verlag Heidelberg, 2004.

[19] Arend Rensink. Towards Model Checking Graph Grammars. In Michael Leuschel, S. Gruner, and S. Lo Presti, editors, *3rd Workshop on Automated Verification of Critical Systems (AVoCS)*, Technical Report DSSE–TR–2003–2, pages 150–160. University of Southampton, 2003.

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

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
63	978-3-86956-204-9	**Theories and Intricacies of Information Security Problems**	Anne V. D. M. Kayem, Christoph Meinel (Eds.)
62	978-3-86956-212-4	**Covering or Complete? Discovering Conditional Inclusion Dependencies**	Jana Bauckmann, Ziawasch Abedjan, Ulf Leser, Heiko Müller, Felix Naumann
61	978-3-86956-194-3	**Vierter Deutscher IPv6 Gipfel 2011**	Christoph Meinel, Harald Sack (Hrsg.)
60	978-3-86956-201-8	**Understanding Cryptic Schemata in Large Extract-Transform-Load Systems**	Alexander Albrecht, Felix Naumann
59	978-3-86956-193-6	**The JCop Language Specification**	Malte Appeltauer, Robert Hirschfeld
58	978-3-86956-192-9	**MDE Settings in SAP: A Descriptive Field Study**	Regina Hebig, Holger Giese
57	978-3-86956-191-2	**Industrial Case Study on the Integration of SysML and AUTOSAR with Triple Graph Grammars**	Holger Giese, Stephan Hildebrandt, Stefan Neumann, Sebastian Wätzoldt
56	978-3-86956-171-4	**Quantitative Modeling and Analysis of Service-Oriented Real-Time Systems using Interval Probabilistic Timed Automata**	Christian Krause, Holger Giese
55	978-3-86956-169-1	**Proceedings of the 4th Many-core Applications Research Community (MARC) Symposium**	Peter Tröger, Andreas Polze (Eds.)
54	978-3-86956-158-5	**An Abstraction for Version Control Systems**	Matthias Kleine, Robert Hirschfeld, Gilad Bracha
53	978-3-86956-160-8	**Web-based Development in the Lively Kernel**	Jens Lincke, Robert Hirschfeld (Eds.)
52	978-3-86956-156-1	**Einführung von IPv6 in Unternehmensnetzen: Ein Leitfaden**	Wilhelm Boeddinghaus, Christoph Meinel, Harald Sack
51	978-3-86956-148-6	**Advancing the Discovery of Unique Column Combinations**	Ziawasch Abedjan, Felix Naumann
50	978-3-86956-144-8	**Data in Business Processes**	Andreas Meyer, Sergey Smirnov, Mathias Weske
49	978-3-86956-143-1	**Adaptive Windows for Duplicate Detection**	Uwe Draisbach, Felix Naumann, Sascha Szott, Oliver Wonneberg
48	978-3-86956-134-9	**CSOM/PL: A Virtual Machine Product Line**	Michael Haupt, Stefan Marr, Robert Hirschfeld
47	978-3-86956-130-1	**State Propagation in Abstracted Business Processes**	Sergey Smirnov, Armin Zamani Farahani, Mathias Weske
46	978-3-86956-129-5	**Proceedings of the 5th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering**	Hrsg. von den Professoren des HPI

ISBN 978-3-86956-217-9
ISSN 1613-5652