On Safe Service-Oriented Real-Time Coordination for Autonomous Vehicles[*]

Basil Becker Holger Giese

System Analysis and Modeling Group
Hasso Plattner Institute at the University Potsdam
Professor-Doktor-Helmert-Str. 2-3, 14482 Potsdam, Germany
E-mail: [Basil.Becker|Holger.Giese]@hpi.uni-potsdam.de

Abstract

The performance of autonomous vehicles could be drastically improved if ad-hoc networking and suitable real-time coordination is employed to optimize and improve the joint behavior of multiple autonomous units. However, due to ad-hoc connections and the real-time interaction the correctness and safety of such coordinated autonomous units is very hard to ensure. In this paper we present how service-oriented real-time coordination can be employed to achieve this goal. Based on the proper real-time coordination between two or more vehicles captured by a service contract, we focus on structural changes and the instantiation and termination of service contracts which is a crucial prerequisite for a safe system operation. We present how the structural changes and the service contract creation/deletion can be modeled by a well-defined UML subset consisting of class and object diagrams with collaborations as well as well-defined behavioral rules can be verified taking the dynamic structural changes due to the ad-hoc networking as well as the real-time coordination into account. The new verification technique is outlined and the application of the technique for an application example is presented.

1. Introduction

If ad-hoc networking and suitable real-time coordination is employed to enable a coordinated behavior of the autonomous units, their performance could be drastically improved (cf. [15]). 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. [13]).

Traditional modeling approaches such as component-based modeling do not cover the dynamic character of the sketched coordination concepts well. Therefore we suggest using instead a service-oriented approach, which employs collaborations of multiple roles in form of service contracts (cf. [1, 6]). We present how such service-oriented view can be modeled by a well-defined subset of UML class diagrams with collaborations as well as well-defined behavioral rules for the structural changes and service contract instantiation or termination.

Due to the safety-critical character of most autonomous vehicle coordination tasks, means for their proper verification are necessary. The formal verification of the coordination of vehicles and variables structures is already a major challenge for a fixed finite set of vehicles and a finite topology [8, 3, 18, 16, 13]. If also time is considered, models such as timed automata or timed Petri net may be employed. However, due to the time aspect the state space of the considered finite models often is too large for verification. In [10], we present our approach to tackle this problem. We employ model checking to verify the real-time behavior of the interaction of a collaboration and its roles separately. We then combine this in a compositional manner with the verification of the component synchronization by ensuring that the components refine the collaboration roles.

In another work [5], we developed an approach to address the formal verification of systems with dynamical structural adaptation as it occurs for example when autonomous vehicles build convoys. There, a technique to automatically check inductive invariants for potentially infinite state models for system with autonomous vehicles and their active collaborations for pure structural rules without time has been presented.

The proper combination of the former approach to verify the real-time coordination for one collaboration and the outlined pure structural rules has been presented in [9]. However, the outlined approach is limited to solutions where the structural rules which describe the system behavior as well as the creation or deletion of collaborations can be proven to be safe even though timing is not considered. This seriously

[*]This work was partly developed in the course of the Collaborative Research Center 614 – Self-optimizing Concepts and Structures in Mechanical Engineering – University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

restricts the covered set of solutions and usually only covers solutions which perform less optimal than those which employ time related behavior. In addition, models which support time can cover the essence of the real-time coordination problem more naturally and thus are usually much easier to model than attempts which ”encode” the same problem using a model with pure structural rules.

In this paper we therefore present how the correct instantiation and termination of service contracts which is a crucial prerequisite for a safe system operation can be modeled and also guaranteed including the timing. A new verification technique extending the results of [5] is presented which supports also structural adaptation rules with time constraints. Using the ideas of [9], we can combine the new technique to guarantee the existence of safe real-time coordination between two or more vehicles captured by a service contract with the compositional real-time model checking of the collaborations [10] to ensure the overall safety.

The paper is structured as follows: We first review the current state-of-the-art for approaches tackling the verification of real-time coordination of autonomous vehicles 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 outline the underlying formal model in Section 4. Our new approach to verify crucial safety properties for such timed models is presented in Section 5. The results of the verification of the considered application example are sketched at the end of Section 5 before we finish the paper with a conclusion and outlook on future work.

2. State of the Art

A number of related approaches for the verification of systems with structural changes like our earlier work [5] exist which do not support time dependent behavior: DynAlloy [8] extends Alloy [11] 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 [3] is the one of Varró et al. It transforms visual models based on graph theory into a model-checker specific input [18]. A more direct approach is GROOVE [16] by Rensink where the checking works directly with the graphs and graph transformations. In [13] an untimed 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. 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 [2] 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 [4] 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 [14] 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.

3. Modeling

The rail-cab[1] project that will serve as application example aims at developing a new intelligent transportation system that is based on small and autonomous shuttles. The shuttles are planned to replace the existing railway systems by providing small transportation units – the shuttles – that can be booked by single persons or small groups on demanded and reach the destination without the need to change the shuttle. In order to minimize energy consumption the shuttle build convoys where thus a safe real-time coordination among the shuttles is crucial.

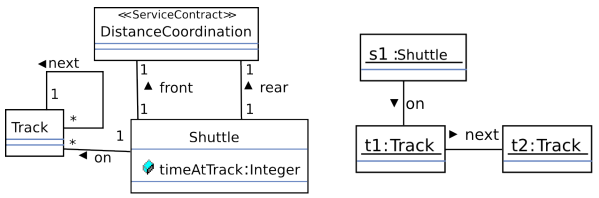

Figure 1. UML class diagram describing the system’s elements and their relation to each other together with an object diagram

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

The applicability of services for the real-time and automotive domain has been, for example, demonstrated in [6], where services are introduced as first-class concept for the collaboration among components. Following this definition and the proposal for modeling service contracts with UML at the OMG supported by many main tool vendors [1], we suggest to use UML to model the system's entities and service contracts in form of collaborations to model the real-time coordination between the autonomous units.

Structure. UML class diagrams are used to model the set of possible states of our system. In Figure 1 the class diagram we used as ontology is shown. Our system consists of Shuttles that can be located at Tracks and each Track can have a successor that can be reached via the next association. To coordinate for building a convoy the Shuttles have to establish a collaboration between them that is represented by the DistanceCoordination service contract. If the DistanceCoordination service contract is instantiated between two Shuttles, these two Shuttles are said to collaborate in order to build a convoy. The DistanceCoordination service contract provides two different roles the participants can play - a front and a rear role. The Shuttle that owns the front role is driving in front of the other Shuttle that in turn has to react to the first Shuttle's actions. In this work we focus on the existence of the service contracts and leave out the verification of the collaboration described by the service contract. Service contracts themselves could be verified using our compositional real-time model checking for UML collaborations [10].

Attributes and clocks are modeled as attributes of the objects. To distinguish different attribute instances of different objects, the instances are prefixed with the object's name.

Behavior. The operations of the system we investigate in this paper are modeled by Story Pattern. In Figure 2 such a Story Pattern is shown. Story Pattern are a special extension of UML object diagrams [12] that are able to describe changes to the object structure in a compact fashion. The meaning of the mentioned Story Pattern can be translated as follows: The Shuttle s1 only advances from Track t1 to Track t2 if there is no other Shuttle located at Track t2. Further, this Story Pattern has a guard ($s1.timeAtTrack > 10$) and an update statement ($s1.timeAtTrack' = 0$) attached to it. A more precise and formal definition will be given in Section 4.

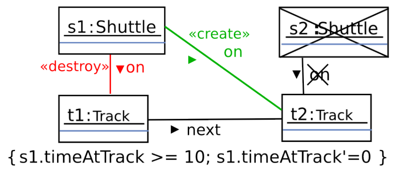

Figure 2. moveSimple rule Story Pattern

In case the DistanceCoordination service contract is instantiated between two Shuttles, it is safe for two Shuttles to be located at the same Track. The Story Pattern in Figure 3 shows the corresponding rule. The last two rules of the system we require is the creation and deletion of the DistanceCoordination service contract. These rules are depicted in Figure 4 and 5.

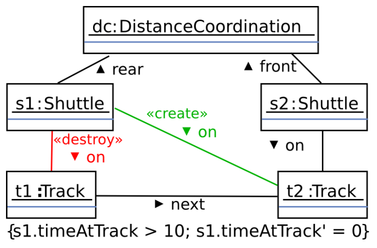

Figure 3. moveDC rule Story Pattern

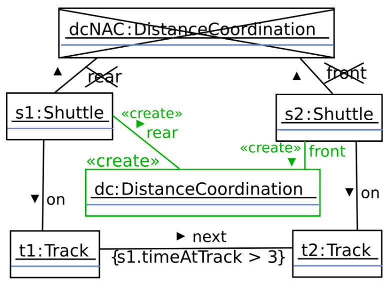

Figure 4. createDC rule Story Pattern

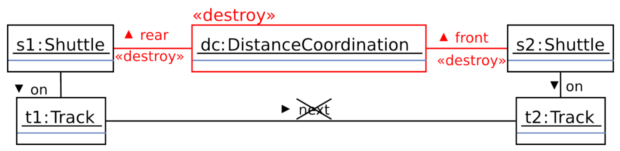

Figure 5. deleteDC rule Story Pattern

The more natural modeling resulting from the support of time leads to a reduction of the number of rules needed to model the system – 6 rules in [5] compared to 4 in this work. The difference become even more apparent when you look at the average size of the rules for the timed model which is only half of the size required in [5].

Safety Properties. To guarantee the safety of the autonomous vehicles, we have to exclude states which represent an accident or a hazard. Therefore, we use Story Pattern without creation or deletion of elements as well as updates to describe instance situations which we consider to be hazards.

For our application example there is essentially only one hazard: the collision of shuttles. We capture this by two forbidden cases. The first one describes a potential collision between two Shuttles which are located on the same track

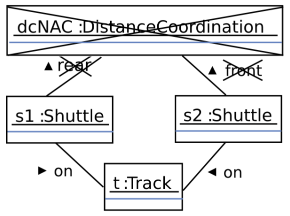

Figure 6. Forbidden Story Pattern collision

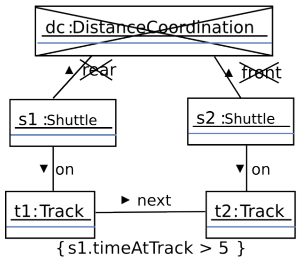

Figure 7. Forbidden Story Pattern noDC

and is depicted in Figure 6 and the second one – shown in Figure 7 – where both shuttles are on connected tracks and the rear one is on that track for at least 5 ms. In both cases the situation is only critical, if a DistanceCoordination service contract between both Shuttles in close proximity is missing.

4. Formal Model

We used class diagrams, object diagrams, and Story Patterns in the preceding section to define possible system states, possible system transitions, and required system properties. The formal model underlying the employed UML subset and Story Patterns - a timed extension of graph transformation systems - is introduced in this section to define the formal semantics and to enable the later considered verification of the properties.

Graphs. A system state, given as an object diagram, can be encoded as an attributed graph by modeling objects as nodes with attributes and links as edges. The system model is based on a set $\mathcal{N}$ of nodes, a set $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ of edges, and a set $\mathcal{T}$ of types. The type of a node or edge is defined by the relation $T : (\mathcal{N} \cup \mathcal{E}) \times \mathcal{T}$, i.e., x is of type t if $(x, t) \in T$. A *graph* $G = (N, E)$ consists of a set $N \subseteq \mathcal{N}$ of nodes and a set $E \subseteq N \times N$ of edges.

The attributes of nodes are based on a set $\mathcal{A}$ of attributes. For each type $t \in \mathcal{T}$ exists a set $\mathcal{A}_t$ of attributes which are available for instances of this type. A *attributed graph* is then a pair (G, α) consisting of a graph $G = (N, E)$ with node set N and a partial function $\alpha : N \times \mathcal{A} \to \mathbb{R}$ providing evaluations. $\alpha(n, a)$ must then be defined for all $n \in N$, $(n, t) \in T$, and $a \in \mathcal{A}_t$. For a condition ϕ over the variables of an evaluation function α, we denote the resulting interpretation as $[[\phi]]_\alpha$.

Consider the set $\mathcal{N} = \{s1, t1, t2\}$ of nodes, the set $\mathcal{E} = \mathcal{N} \times \mathcal{N}$ of edges, the set $\mathcal{T} = \{\mathit{Shuttle}, \mathit{Track}, \mathit{on}, \mathit{next}\}$ of types, and the type relation T with $T = \{(s1, \mathit{Shuttle}), (t1, \mathit{Track}), (t2, \mathit{Track}), ((t1, s1), \mathit{on}), ((t1, t2), \mathit{next})\}$. Then $G = (\mathcal{N}, \{(t1, s1), (t1, t2)\})$ is the graph representing instance diagram depicted in Figure 1.

Graph Patterns. A *graph pattern* $P = (N^+, N^-, E^+, E^-)$ consists of two sets N^+ and N^- of positive and negative nodes, and two sets $E^+ \subseteq N^+ \times N^+$ and $E^- \subseteq (N^+ \cup N^-) \times (N^+ \cup N^-)$ of positive and negative edges. We further use P^+ to denote P restricted to N^+ and E^+. A graph pattern represents the set of graphs that match the pattern. A graph G *matches* a graph pattern P if there exists an isomorphic function m that maps the positive nodes and positive edges of P to nodes and edges of G, respectively, and m cannot be extended in such a way that it matches any negative node or negative edge of P to a node or edge in G, respectively. The matching function m preserves types, i.e., a node or edge may only be mapped to a node or edge of the same type, respectively.

An *attributed graph pattern* is a pair (P, ϕ_P) consisting of a graph pattern P and a condition ϕ_P over the attributes of the nodes in N^+. An attributed graph pattern matches an attributed graph (G, α) if P matches G for an isomorphism iso and the condition is also true for ϕ_p ($[[\phi_P]]^{iso}_\alpha = true$), where $[[\phi]]^{iso}_\alpha$ denotes the interpretation for α where the nodes are mapped according to iso.

A graph pattern P *matches* another graph pattern P' if there exists an isomorphic function iso that maps all positive elements of P to positive elements of P' and all negative elements of P to negative elements of P'. If P matches P', we say that P is a subpattern of P', and write $P \sqsubseteq P'$. For an attributed graph pattern (P, ϕ_P) and $(P', \phi_{P'})$ we require for a match that besides the graph matching that the condition over the attributed holds $\forall \alpha : [[\phi_{P'}]]^{iso}_\alpha \Rightarrow [[\phi_P]]^{iso}_\alpha$ and we write $(P, \phi_P) \sqsubseteq (P', \phi_{P'})$.

Both kinds of graph patterns that are used to describe system properties can be divided into required and forbidden patterns. A *required* pattern must always be fulfilled during system execution (system invariant), whereas a *forbidden* pattern must never be fulfilled (system hazard, accident).

Graph Transformation Rules. An attributed graph transformation transforms one graph into another one by creating new graph elements (nodes or edges), updating attribute values for the nodes, and removing existing graph elements. Transformation rules define sets of possible graph transformations. If an attributed graph represents the state of a system, an attributed graph transformation represents an update of the system's state, and a sequence of transformations represents an execution of the system.

Story Patterns are extended graph patterns that allow the annotation of graph elements with the stereotypes ≪create≫ and ≪destroy≫. As informally introduced above, Story Patterns are used to specify transformation rules. A graph transformation rule $(L, R)_r$ consists of two graph patterns, a left hand side L (LHS) and a right hand side R (RHS). L consists of those elements of the Story Pattern that are not annotated with ≪create≫, including negative elements, whereas R consists of all elements not annotated with ≪destroy≫. The elements annotated with ≪create≫ will be created by the rule (elements from $R^+ \setminus L^+$), while those annotated with ≪destroy≫ will be deleted (elements from $L^+ \setminus R^+$). Elements without annotations are preserved by the application (elements from $L^+ \cap R^+$). For the Story Pattern moveSimple in Figure 2 we thus have that L^+ contains the nodes for s1, t1, and t2 while L^- contains only the node s2.

To also cover attributed graphs, we have attributed graph rules $((L, \phi), R, \mu)_r$ with ϕ a condition over L and μ an evaluation for all $R^+ \setminus L^+$ and an arbitrary subset of $R^+ \cap L^+$ where (L, ϕ) instead of L must be matched and the resulting attribute evaluation is determined by the update μ. In the moveSimple of Figure 2 the condition ϕ equals the guard $s1.timeAtTrack > 10$ whereas μ equals the update statement $s1.timeAtTrack' = 0$.

Graph Transformation Systems. A *graph transformation system* (GTS) $S = (\mathcal{R}, prio)$ consists of a set of graph transformation rules $\mathcal{R}$ (defined by a set of Story Patterns), defining all possible transformations in the transformation system, and a priority function $prio : \mathcal{R} \to \mathbb{N}$, which assigns a priority to each rule (the higher the number assigned to a rule the higher the rule's precedence). An additional set of initial graphs may describe the initial states of the system.

A rule $(L, R)_r$ is *applicable* to a graph G if G matches L *and* G does not match the LHS L' of any other rule $(L', R')_{r'}$ with higher priority. During the *application* of a rule $(L, R)_r$ 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.[2] We write $G \to_r G'$ if rule r can be applied to graph G and the application results in graph G'. We write $G \to^* G'$ if G is transformed into G' by a (possibly empty) sequence of rule applications.

Given any kind of graph transformation system S and a graph G, the set of graphs producible (i.e., the set of states reachable) by applying rules from S to G is denoted by $REACH(S, G) = \{G' \mid G \to^* G'\}$.
An *attributed graph transformation system* (AGTS) $S = (\mathcal{R}, prio)$ consists of a set of attributed graph transformation rules $\mathcal{R}$ and a priority function $prio$.

An attributed rule $((L, \phi), R, \mu)_r$ is *applicable* to an attributed graph (G, α) if G is matched by $(L, R)_r$, ϕ evaluates to true for α, and (G, α) does not match any other rule with higher priority. We write $(G, \alpha) \to_r (G', \beta)$ if rule r can be applied to the attributed graph (G, α), $G \to_{r'} G'$ for $(L, R)_{r'}$ and β the evaluation which results from μ.

[2]Our notion of application is a restricted version of the *Single Pushout* approach (cf. [17]). For more details please see [5].

Timed Graph Transformation Systems. To also cover timed behavior we decompose the set of attributes into a set $\mathcal{C} \subseteq \mathcal{A}$ of clocks and a set of normal attributes $\mathcal{A} \setminus \mathcal{C}$. Given an evaluation function α, we refer to that evaluation function where the values of all clock variables is incremented by x as $\alpha \oplus x$ (resp. $\alpha \ominus x$ for decrement). We can transfer both definitions to conditions over the normal attributes and clocks by substituting appropriate offsets.

A *timed graph transformation system* (TGTS) $S = (\mathcal{R}, \mathcal{R}_u, prio)$ is then defined for $(\mathcal{R}, prio)$ a valid attributed graph transformation system and $\mathcal{R}_u \subseteq \mathcal{R}$ the subset of all urgent rules.

In the case of TGTS, we have additional *time steps* where the clock values increase over time. For $\delta \geq 0$ we have $(G, \alpha) \to_\delta (G', \alpha \oplus \delta)$ if for all $x \leq \delta$ holds $(G, \alpha \oplus x)$ does not match any urgent rule $r' \in \mathcal{R}_u$. We write $(G, \alpha) \to^* (G', \beta)$ if (G, α) is transformed into (G', β) by a (possibly empty) sequence of rule applications and time steps.

For an attributed or timed graph transformation system S and an attributed graph (G, α), the set of attributed graphs producible (i.e., the set of states reachable) by applying rules is denoted by $REACH(S, (G, \alpha)) = \{(G', \beta) \mid (G, \alpha) \to^* (G', \beta)\}$.

5. Checking

Given UML models describing the service-oriented real-time coordination of autonomous vehicles as outlined in Section 3 and their underlying formal model in form of TGTS, we present in this section our verification technique which covers the time dependent structural changes as well as instantiation and termination of the service contracts. We will first review the basic idea of our former approach [5] for untimed models and then describe how we extend it to also cover time dependent behavior. Finally, we present the results for the checking of the complete example.

Untimed Case. In our approach a set of forbidden graph patterns $\mathcal{F} = \{F_1, \ldots, F_n\}$ are employed to represent possible safety-violations (hazards, accidents) of our system. The related property $\Phi_\mathcal{F}$, denoted by $G \models \Phi_\mathcal{F}$, holds iff G matches none of the graph patterns in $\mathcal{F}$. We call G a *witness* for the property $\neg\Phi_\mathcal{F}$ if G in contrast matches a forbidden graph pattern $F \in \mathcal{F}$.

The property $\Phi_\mathcal{F}$ is an *operational invariant* of the GTS S iff for a given initial graph G^0 for all $G \in \mathsf{REACH}(S, G^0)$ holds $G \models \Phi_\mathcal{F}$ (cf. [7]). However, due to the Turing-completeness of graph transformation systems with types checking them is restricted to finite models and thus does not fit to the considered class of problems. We therefore instead tackle the problem whether the property $\Phi_{\mathcal{F}}$ is an *inductive invariant*. This is the case if for all graphs G and for all rules $r \in \mathcal{R}$ holds that $G \models \Phi_{\mathcal{F}} \wedge G \rightarrow_r G'$ implies $G' \models \Phi_{\mathcal{F}}$. If we have an inductive invariant and the initial graph G^0 fulfills the property, then $\Phi_{\mathcal{F}}$ is also an *operational invariant* as inductive invariants are stronger than their operational counterparts.

We can reformulate the definition of an *inductive invariant* as follows to have a falsifiable form: a property $\Phi_{\mathcal{F}}$ is an inductive invariant of a GTS $S = (\mathcal{R}, prio)$ if and only if there exists no pair (G, r) of a graph G and a rule $r \in \mathcal{R}$ such that $G \models \Phi_{\mathcal{F}}$, $G \rightarrow_r G'$ and $G' \not\models \Phi_{\mathcal{F}}$. Such a pair (G, r) which witnesses the violation of property $\Phi_{\mathcal{F}}$ by rule r is then a *counterexample* for the initial hypothesis.

As explained in detail in [5], we can exploit the fact that the application of a rule can only have a local effect to verify whether a counterexample exists. A counterexample (G, r) can only exist when the local modification of G by rule r is necessarily responsible for transforming the correct graph G into a graph that violates the property. In addition, to have a possible counterexample we require that the rule r is not preempted by a rule r' with higher priority.

As we can represent the infinite many possible counterexamples by an only finite set of representative patterns $\Theta(R_l, F_i)$ of graph patterns P' that are combinations of a RHS R_l of a rule r_l and a forbidden graph pattern $F_i \in \mathcal{F}$ (cf. [5]), we can check that no counterexample exists (and $\Phi_{\mathcal{F}}$ is thus an inductive invariant) only considering this finite set.

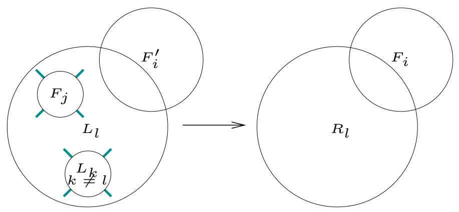

Figure 8. Schema to check a potential counterexample (P, r_l) with resulting graph pattern P' that is a combination of a RHS R_l of a rule r_l and a forbidden graph pattern $F_i \in \mathcal{F}$

As depicted in Figure 8 we have to check for any graph pattern $P' \in \Theta(F_i, R_l)$ for some $F_i \in \mathcal{F}$ and $r_l \in \mathcal{R}$ whether the pair (P, r_l) with P defined by $P \rightarrow_r P'$ is a counterexample for $\Phi_{\mathcal{F}}$ or not as follows:

1. Check that the rule r_l can be applied to graph pattern P and that the resulting graph pattern is P' (this implies that no $r_k \in \mathcal{R} \backslash \{r_l\}$ exists with $prio(r_k) > prio(r_l)$ that matches P, due to the definition of rule application).

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

We use the above conditions in our algorithm to check if a counterexample exists. The algorithm performs this check for any given rule $(L, R)_r \in \mathcal{R}$ and forbidden graph pattern $F \in \mathcal{F}$. The algorithm therefore computes the set of all possible target graph patterns for R and the forbidden graph pattern F $(\Theta(R, F))$ and then derives the related source graph patterns. The above sketched conditions are then checked for all source graph pattern to decide whether the source graph pattern P represents potentially safe graphs that can be transformed into unsafe graphs by applying r. If so, the pair (P, r) is a valid counterexample.

We have demonstrated in [5] that this problem can be tackled algorithmically with an explicit as well as symbolic algorithm. However, the solution developed so far did not support time dependent behavior.

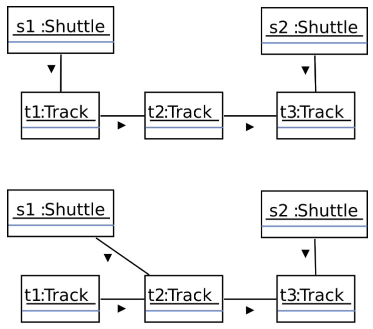

Figure 9. Witness against systems correctness using the untimed verification

To exemplify the verification algorithm we consider the following situation: L_k is moveSimple (see Figure 2) and F_i is noDC (see Figure 7). In Figure 9 a possible pair of source- and target graph pattern is shown. It could easily be seen that the depicted target graph pattern is a combination of moveSimple and noDC. As the source graph pattern does not contain any forbidden subgraph the GTS is not safe. Due the fact the clock constraints are not considered, we have this negative result.

Timed Case. The extension of the modeling technique for time dependent behavior as outlined in Section 3 changes our checking problem considerably. In the employed timed model, the behavior is described by rule applications and time steps. Therefore, reaching a forbidden graph pattern in principle could involve a rule application as well as a time step. The basic idea to approach the checking is to extend the untimed case by mapping the time related aspects on a system of linear inequalities which can then be checked by a constraint solver which require that all conditions of the timed graph transformation system are linear.[3]

We can analogously to the untimed case formulate the

[3]It is to be noted that also early versions of UPPAAL (cf. [19]) relied on constraint solving for the verification of real-time systems.

definition of an *inductive invariant* for the timed case in a falsifiable form: a property $\Phi_{\mathcal{F}}$ with forbidden attributed graph pattern $(F_i, \psi_i) \in \mathcal{F}$ is an inductive invariant of a TGTS $S = (\mathcal{R}, \mathcal{R}_u, prio)$ if and only if there exists no pair $((G, \alpha), r)$ of an attributed graph (G, α) and an attributed rule $r \in \mathcal{R}$ such that $(G, \alpha) \models \Phi_{\mathcal{F}}$, $(G, \alpha) \rightarrow_r \rightarrow_\delta (G', \beta)$, 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 timed case.[4] Using the same idea as for the untimed case we can lift this problem to attributed graph pattern. Again, only a 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 $r_l = ((L_l, \phi_l), R_l, \mu_l)_{r_l}$ and a forbidden graph pattern $(F_i, \psi_i) \in \mathcal{F}$ have to be considered.

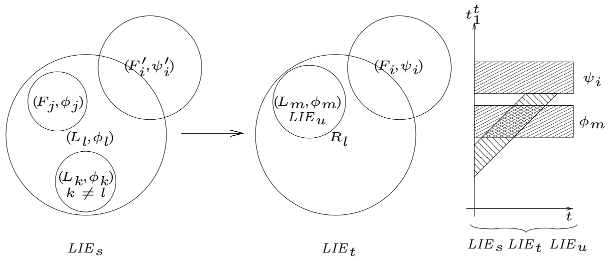

Figure 10. Schema to check a potential counterexample $((P, \phi_P), r_l)$ with resulting graph pattern $(P', \phi_{P'})$ that is a combination of a RHS R_l of a rule r_l and a forbidden graph pattern $(F_i, \psi_i) \in \mathcal{F}$ in the timed case

As depicted in Figure 10 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, ϕ_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 attributed graph pattern (P, ϕ_P) and that the $(P', \phi_{P'})$ results from this application plus a time step of length $\delta \geq 0$ (this implies 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 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 that there exists no $(F_j, \phi_j) \in \mathcal{F}$ with $(F_j, \phi_j) \sqsubseteq (P, \phi_P)$ (otherwise (P, ϕ_P) is already invalid).

Therefore, the extended checking algorithm employs in its first step a slightly adjusted version of the untimed algorithm to derive potential counterexamples (see Figure 10). 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 clock constraints that a match found in the source graph pattern does not directly invalidate the counterexample but rather restrict the possible clock values. We derive a system of linear inequalities LIE_s to encode which values for $t^s_1, \ldots, t^s_n$ 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, ϕ_P) and $iso'(\phi_k)$ for all matches iso' of L_k in (P, ϕ_P).

Concerning the application of the rule r, we have to take into account that the rule may either not affect clock variables (which are therefore in their possible values determined by LIE_s) or update them to a given constant. While in the untimed case the match of the forbidden graph pattern in the target graph pattern is sufficient to ensure that $\Phi_{\mathcal{F}}$ is not valid any more, in the TGTS case the related forbidden attributed graph pattern (F_i, ψ_i) may also require that the clock variables fulfill the additionally specified conditions (see the ψ_i area in the constraint space on the right-hand-side of Figure 10). We therefore encode the continuous evolution after the discrete step in an additional system of linear inequalities LIE_t with a special variable t and describe all not updated clocks in the target graph pattern as $t^t_i = t^s_i + t$ and the ones updated to c_i as $t^t_i = c_i + t$.

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 timed case urgent rules may in fact prevent that we reach a clock evaluation which fulfills the clock constraints of the structural embedded forbidden graph patterns. This effect is encoded in a third system of linear inequalities including boolean conditions LIE_u (see the ϕ_m area in the constraint space on the right-hand-side of Figure 10). The system of linear inequalities LIE_u defines upper bounds for the special variable t. These upper bounds could be computed from the conditions of each embedded urgent rule. Assuming the condition ϕ_r is an urgent rule's condition we solve it for the special variable t and hence yield an interval of points in time when the urgent rule r is activated. As urgent rules have to be implied as soon as they are applicable only the lower bounds of this interval are of interest. These lower bounds will be used as the upper bound for the special variable t in LIE_u.

Thus, we can finally use the combination of the systems of linear inequalities LIE_s, LIE_t, and LIE_u to check that the considered structural counterexample is also a counterexample in the timed model.

We use the same setup as in untimed case to perform the verification for the timed case. In addition to the structural analysis the linear equation system has to be feasible. For this small example the equation system is: $s1.timeAtTrack \geq 10 \wedge 0 + t \geq 5$ which is obviously feasible. The system could thus not be considered to be safe. A inspection of the counterexample reveals that the

[4]This condition in fact requires that for an initial state (G, α) we check not only $(G, \alpha) \models \Phi_{\mathcal{F}}$ but also $(G, \alpha \oplus x) \models \Phi_{\mathcal{F}}$ for all x with $(G, \alpha) \rightarrow_x (G, \alpha \oplus x)$.

createDC rule (see Figure 4) is not excluding this case as the rule was erroneously not defined as urgent.

Correcting the identified problem by defining the timed graph transformation createDC as urgent, the linear equation system is augmented with the additional constraint $0 + t \leq 3$. This resulting linear equation system is unfeasible, hence the pair shown in Figure 9 is not a witness against the system's correctness anymore.

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.

We have used our algorithm to finally check that no witness can exist for the corrected timed model which thus has been proven to be safe (does not exhibit any forbidden graph pattern). The verification of the whole system for the timed model lasts in average 329 ms, ca. 40 % of the time has been spent for solving of the linear equation systems.[5] During verification the algorithm expanded the 8 possible combinations of graph rules and forbidden graph patterns to 68 pairs of graph pattern.

If one compares the number of 6 graph transformations and 15 forbidden subgraphs that were necessary for modeling the application example with an untimed model in [5] with the 4 graph transformations and 2 forbidden subgraphs we require for the timed model, the benefits of explicitly modeling time with clocks and urgent graph transformations become obvious.[6]

6. Conclusion and Future Work

We presented how the safe creation and deletion of service contracts modeled by class diagrams, collaborations, and rules for the instantiation and termination of the collaborations can be verified. The presented verification techniques does not only address the dynamic structural changes due to the ad-hoc networking using graph transformation theory but also takes the real-time behavior of the rules into account. Therefore, the presented approach in contrast to former work in the area can also address solution where the proper operation requires time related behavior. Analogously to the arguments in [9] how to combine the untimed approach [5] with the verification of the collaborations in [10], we can employ the presented technique for timed models to ensure the overall safety.

In the future we plan to investigate several directions: going back not only a single but multiple steps and modeling a checking support for clock drift.

[5]HW configuration: Intel P4 2.8GHz, 512MB running Linux 2.6.24

[6]The dramatic decrease for the number of forbidden graph patterns from 15 to 2 has also be achieved by encoding some cardinality constraints given by the class diagram within the structural checks themselves rather than using additional forbidden subgraphs.

References

[1] 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.

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

[3] 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.

[4] J. Bauer and R. Wilhelm. *Static Analysis of Dynamic Communication Systems by Partner Abstraction*, chapter 16, pages 249–264. LNCS 4634. Springer Berlin / Heidelberg, 2007.

[5] B. Becker, D. Beyer, H. Giese, F. Klein, and D. Schilling. Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation. In *Proc. ICSE*, pages 72–81. ACM Press, 2006.

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

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

[8] M. F. Frias, J. P. Galeotti, C. L. Pombo, and N. Aguirre. DynAlloy: Upgrading Alloy with actions. In *Proc. ICSE*, pages 442–451. ACM, 2005.

[9] H. Giese. Modeling and verification of cooperative self-adaptive mechatronic systems. In *Reliable Systems on Unreliable Networked Platforms*, LNCS 4322, pages 258–280. Springer Verlag, 2007.

[10] H. Giese, M. Tichy, S. Burmester, W. Schäfer, and S. Flake. Towards the compositional verification of real-time uml designs. In *Proc. ESEC*, pages 38–47. ACM Press, September 2003.

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

[12] H. Köhler, U. Nickel, J. Niere, and A. Zündorf. Integrating UML diagrams for production control systems. In *Proc. ICSE*, pages 241–251. ACM, 2000.

[13] 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.

[14] 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.

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

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

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

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

[19] W. Yi, P. Pettersson, and M. Daniels. Automatic verification of real-time communicating systems by constraint-solving. In D. Hogrefe and S. Leue, editors, *FORTE*, volume 6 of *IFIP Conference Proceedings*, pages 243–258. Chapman & Hall, 1994.