Towards Safety Guarantees for Service-oriented Systems

Basil Becker
System Analysis and Modeling Group
Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmertstr. 2-3, D-14482 Potsdam, Germany
Basil.Becker@hpi.uni-potsdam.de

Abstract

In this paper I will address the problem of verifying that a service-oriented embedded system fulfills required safety guarantees. Services will be represented by collaborations among components with a varying number of participants. The services are equipped with rules for the instantiation and deletion of services, addition and removal of participants. Further a service’s behavior could be modified at run-time. I will present techniques which are able to verify that such systems are safe with respect to a given specification. The sketched verification technique is able to cope with structural changes, run-time changes of rules and the run-time evolution of services.

1. Introduction

It is a widely observed fact that the complexity of software systems is constantly increasing. In the same amount the complexity increases computer scientists develop new abstraction techniques. The development some years ago for the time being reached its peak with the emergence of the paradigm of service-oriented architecture (SOA).

The term SOA is mostly associated with business computing and within this domain it is associated with the properties: loosely coupled, late binding and decentralized. Thereby decentralized characterizes the crossing of organizational borders, late binding refers to the fact that communication partners are determined at run-time and not at compile-time and loosely coupled describes that the bindings could be volatile. All of these three properties could easily be found in embedded software-intensive systems, such as car-to-car communication systems, systems using mobile phones, systems that have to cope with unreliable communication and autonomous vehicles just to name a few. Thus, it is not surprising that in the last years services have also been applied to the domain of embedded systems (cf. [9]).

Most of the systems mentioned above contain at least some safety-critical parts. Hence, special attention has to be paid to the modeling of such systems. Testing is not able to sufficiently guarantee the systems’ correctness, thus formal verification techniques have to be employed. The changes occurring at the service level are structural changes and the verification as well as the modeling technique have to be able to express those changes. For embedded and safety-critical systems time related behavior is important and has to be supported by the verification and modeling technique.

Throughout this paper I will use a real-life application example to illustrate the outlined problems, concepts and solutions. The RailCab project at the University of Paderborn has developed a new railway technology, which is a combination of public and individual transportation. Small autonomous transportation units, called Shuttles, could be ordered by customers via the Internet or the customers’ mobile phone. The Shuttles bear the problem of high energy consumption per passenger. To reduce it the Shuttles can build convoys, what requires small distance between the Shuttles together with high velocities and following a safe coordination between succeeding Shuttles. This coordination is encoded in a protocol running between two succeeding Shuttle. The protocol is referred to as distance coordination collaboration - a service of the RailCab system (cf. [12]).

2. Problem Statement

The problem my thesis addresses is to bring the dynamic aspects of service-oriented systems to the domain of embedded software-intensive systems. Therefore, all changes, which occur in a service-oriented architecture, have to be explicitly modeled. I will first depict a useful and accepted translation of services into the domain of software-intensive, embedded systems. Building on top of this definition I will clarify what kind of dynamics can occur in

1http://www.railcab.de
such a system. I will investigate structural changes, changes of the collaborations modification rules and the modification of collaborations at run-time. The last class of changes could be partitioned into compatibility preserving and compatibility destroying ones. Of course, the correctness of the collaborations themselves has to be verified, too. The explicit modeling of possible changes in the system allows me to give a sketch of the verification to be performed for each of them.

In this context a service is a reusable interaction scheme, defined over a set of roles, with a potentially varying number of interacting roles (cf. [9, 1]). We use standard UML component diagrams for the modeling of components. Each component can implement multiple roles, which were defined in terms of UML interfaces. Given this, a service is modeled by an UML collaboration diagram, which contains the roles that can participate in the service (cf. [1]). An automaton models the roles’ behavior. Components involved into a service instance are called participants, the terms service and collaboration can be used interchangeably. Instantiation and destruction of collaborations as well as joining and leaving of participants is modeled by a set of so called modification rules, which could formally be interpreted as typed and attributed graph transformations. The mapping of rules to graph transformation is the formal foundation for all verification techniques used in this paper.

3. Proposed Solution

Each verification technique needs a formal specification the verified system has to fulfill. Therefore, it is required that the system designer explicitly specifies, which states are forbidden. This is done through a set of safety properties \( \mathcal{F} \).

3.1. Structural Changes

Structural changes in the context of service-oriented systems means that either a new collaboration instance has been created, a running collaboration has been removed, a participant has joined a running collaboration or a participant has left a collaboration. To make a long story short: a modification rule has been applied to the system. It has to be noted that structural changes also imply behavioral changes. Each of the possible structural changes can be described by a rule. Applied to the application example the creation and destruction of distance coordination collaborations are good examples for the structural changes that dynamically appear within such a service-oriented system.

Given a set of rules \( \mathcal{R} \), it has to be verified that any application of the rules can’t reach a state \( s \), which violates a safety property \( p_f \in \mathcal{F} \). Therefore we verify that the non-existence of a forbidden state is an inductive invariant of the rule set \( \mathcal{R} \). As the verification result depends on the rule set and the safety properties, only, the result holds for any system, which uses the same set of rules and whose initial state is safe. The technique is called Invariant Checking and is explained in depth in [5].

Accordingly, the verification result remains valid as long as the rule set does not change and the verification of structural changes has to be done only once before the system is deployed. The changes described in this section are modeled and verified at design-time.

3.2. Modification Rule Changes

Structural changes are sufficient for a service-oriented system as long as the available modification rules and following the possible structural changes meet the requirements. But especially in long running systems the specified requirements may change over time. E.g. a new law requires Shuttles to instantiate the distance coordination collaboration earlier. This legal change will result in a change of the system’s safety properties and finally in the need to partially change the modification rules at run-time, to fulfill the changed safety properties again.

Changing the rule set means that earlier verification results are invalid afterwards and the system has to be verified again. Further the new rules have to be distributed among all system components. In a distributed system this could not be assumed to happen atomically and thus the system will - at least for some time - be in a mixed mode, where old and new rules exist in parallel.

Whatever the behavioral changes’ source is, the verification of the new system looks the same. Given the old rule and property set \( \mathcal{R}’ \) and \( \mathcal{F}’ \) and the new rule and property sets \( \mathcal{R} \) and \( \mathcal{F} \) it has to be verified that: (1) \( \mathcal{R} \) satisfies \( \mathcal{F} \cup \mathcal{F}’ \) (2) \( \mathcal{R} \cup \mathcal{R}’ \) satisfies \( \mathcal{F} \cup \mathcal{F}’ \). If the new rule set passes both steps it is safe to distribute the new rule set. Otherwise the planned changes have to be reordered until the verification succeeds.

3.3. Changing Collaborations

Changing a collaboration at run-time is a difficult task as collaborations are often required to permanently exist in the system to guarantee the system’s safety. As aforementioned a changed collaboration can be compatible to the originating one or not. A collaboration \( C’_a \) being compatible to a collaboration \( C_a \) means that the behavior of \( C’_a \)’s roles is also a valid behavior for \( C_a \)’s roles. In the application example this could occur if we change the roles to send feedback faster. In this scenario components are not required to update their roles.

The compatibility relation does not hold, if the following change is applied to the application example. The distance
coordination service, which initially incorporates two succeeding Shuttles only, is replaced by a collaboration, which involves the whole convoy. Such a change at collaboration level requires the change of the roles. In opposite to Section 3.2 we now also change the system’s types.

The collaboration update could be achieved by modifying the modification rules for each collaboration. The modification rules get modified in such a way that each time, when one of them is applied to a collaboration the involved components get updated to be compatible to the new collaboration’s requirements. If components are already using the new collaboration they participate without being updated first. The rules that update the collaboration are called transition rules. The complete process of updating a collaboration from $C_a$ to $C'_a$ can be depicted by these steps: (1) addition of modification rules for the updated collaboration $C'_a$ (2) addition of the transition rules (3) removal of $C_a$’s modification rules (4) removal of transition rules after updating all components. Each of these four steps must not violate the system’s safety. Hence, we need to verify that each of the different rule sets is safe w.r.t. the system’s safety properties.

The transition rules can be removed from the system’s rule set when all old roles have been updated. It has to be guaranteed that this happens after a finite number of transition rule applications. Each application of a transition rule decreases the number of old roles. As the old collaboration’s modification rules are removed from the system, only transition rules can be applied to collaboration instances incorporating old roles. This requires that each collaboration instance gets modified at least once. In case a collaboration instance does not change its structure, it does not get updated. To avoid this and following the permanent mixture of old and updated collaborations, one could implement a system where each component is only allowed to use a certain role if it has been granted a usage permit. Each usage permit is valid for a certain usage period only. Has the usage period passed the component has to renew its usage permit for the role. If the collaboration and so the role has been updated meanwhile, the usage permit is only granted for the new role. Leasing software for a certain period of time is also used in Sun’s Jini\(^2\) technology.

### 3.4. Verified Collaborations

The previous verifications all made the assumption that the used collaborations induce some guarantees and thus their instantiation avoids dangerous situations. I.e. if two Shuttles have the distance coordination service instantiated they can drive in close proximity without colliding. Those assumptions have to be verified. Note that in Section 3.1 the structural changes have been subject to verification, whereas now the focus is on the composed behavior of the roles contained in a collaboration.

For systems where collaborations can only be created and destroyed but participants can’t join or leave in between, the verification can be done using a decompositional approach (cf. [5, 12]). For each collaboration the parallel composition of its roles’ automata is verified using a model-checker. The same is done for the different components together with their implemented roles. In systems – such as those, investigated in this work – where components are allowed to randomly join and leave collaborations the sketched verification approach is not applicable. In general components can join or leave a collaboration in any point in the collaboration’s lifetime. Thus the verification has to reflect this, what leads to a state space explosion.

Currently we are working on a solution incorporating a state space reduction technique for infinite state systems.

### 4. Results achieved so far

The basics for the verification technique used in this work has been introduced in [5]. In [8] the Invariant Checking technique has been extended to also cope with the verification of timed graph transformation systems. In a further publication [6] we have shown in which way Invariant Checking could be extended to work also incrementally. We were able to show the technique’s potential ability to be used for run-time verification tasks. These techniques allow to verify the changes described in Sections 3.1 and 3.2. In [7] the modeling including collaborations and rules has been introduced with a focus on the ability for self-adaptivity.

### 5. Related Work

The techniques presented in this work are able to verify structural changes including changing rule sets and evolution of services. For the class of self-adaptive systems verification of structural changes have been addressed by [18, 13] the employed techniques differ from the presented ideas as they do not support the verification of timed or infinite state systems.

The verification of structural changes has also been investigated in [11, 3, 16, 15]. However, those works does either not support time dependent behavior, require an initial configuration or could only be applied to finite state systems. The possibility to change the rule set during runtime is also not support by these techniques. Only few approaches exist, which also allow for the verification of infinite state systems. [2] transforms graph transformation systems into a combination of a graph and petri net, which then could be analyzed using standard techniques. [4] employs

\(^2\)http://www.jini.org
partner graphs, an abstraction technique, to verify topological properties for the platoon building problem. Both, [4] and [2], are not able to express and verify timed behavior. A tool, capable of verifying real-time behavior with structural changes for finite state systems is Real-Time Maude [14].

In [10] a verification technique for evolving and distributed component based systems is presented, which relies on compositional verification together with predicate abstraction. However, the base algorithm is model checking and thus not suitable for the verification of infinite state systems and structural changes. Further the presented approach does not seem to be applicable to run-time evolution.

In [17] an incremental run-time checking technique for context consistency is presented. This paper differs from my work in the following way as it is only able to detect invariant violations after they have occurred, whereas my approach aims at avoiding unsafe situations by ensuring their non occurrence at run-time.

6. Conclusion

In this paper I have presented my ideas for the verification of safe and evolving service-oriented systems. The tackled problems all originate in the domain of embedded and safety-critical systems, thus verification and analysis of those systems is important. For systems, which employ structural as well as behavioral changes there are already verification techniques available and have been presented at international conferences and workshops (cf. [5, 8, 6]).

Although I already have made some advance towards a solution for the identified problems there are still open issues, which have to be addressed by future work. So is the formalization of the collaboration update a major topic. The same holds for the question whether the changing of collaborations needs special support by modeling notations and under which preconditions an automatic derivation of the transition rules from the changes applied to the collaborations is possible. We have to find a solution for the verification of the collaborations and the presented concepts should be evaluated.

References