Checking Deadlocks in Component Composition with Partial Bindings using Variability Modeling

Vanessa Stricker
(Supervised by Prof. Dr. Klaus Pohl)

paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen, Gerlingstr. 16, 45127 Essen, Germany
vanessa.stricker@paluno.uni-due.de

Abstract. A large number of compositional deadlock checking techniques have been proposed in order to overcome the state-explosion-problem. These approaches require all components in a composition to be fully bound. This restriction conflicts with the underlying reusability paradigm of component-based software engineering, which often requires only parts of components to be reused and allows component compositions to be realized by a large set of similar component configurations. Accordingly, compositional deadlock checking approaches are not applicable to component compositions, which can be realized with partial bindings. We propose to overcome this problem by i) restricting the number of realizations, which have to be verified and ii) removing behavior that is not executed due to partial bindings.

1 Introduction

Component-based software engineering (CBSE) is a well-established paradigm in information system development (especially for distributed information systems), which advocates the reuse of components in order to build a set of large and complex systems from existing components [21]. However, ensuring that components compositions are deadlock free is challenging. Deadlocks are a common source of errors in systems of concurrent processes as they are common in component-based systems [10] because components usually make implicit assumptions about the behavior of their environment. When connecting components, their interaction might give rise to unexpected deviations and inconsistencies w.r.t. the intended behavior of the system. Early verification approaches, like deadlock checking techniques, which are based on the specification of components, have become an important tool in CBSE in order to detect errors in the interaction of components. Unfortunately, deadlock checking of component compositions is subject to the state-explosion-problem, as the concurrent nature of component-based system requires the computation of the global behavior. A large number of compositional deadlock checking techniques have been proposed in CBSE in order to address this challenge. Semmelrock and Majster-Cederbaum [20] were able to prove that compositional deadlock checking is within PSPACE for acyclic component topologies.

1.1 Compositional Deadlock Checking Approaches

Compositional verification generally allows breaking down a verification problem into smaller problems. Under certain conditions, a global property is proven by deduction from local properties of the components in a composition (cf. [4]). Compositional deadlock checking approaches usually concludes the global deadlock freedom of a composition from the local deadlock-freedom of its components. The conditions of most compositional verification approaches require very restrictive composition principles to allow analyzing the smaller verification problems independently from each other. In the case of compositional deadlock checking approaches, these restrictive assumptions require all components to be fully bound, i.e. not to have any partially (cf. for example [1,12]) or unbound ports (c.f. for example [11]).

Two components are bound if there exists a connection between their ports. The components can interact via those connections. If a component has partially bound ports, the interaction of these components within a composition cannot include the unbound functionalities. Thus, the execution of a component within a specific composition might only comprise a subset of the full behavior that has been specified for this component. However, as compositional deadlock checking approaches only analyze small parts of the component's behavior within a composition, they assume that the rest of the composition's behavior adheres to the specified (full) behavior.

1.2 Problem: Unrealistic Assumptions

As the conditions, under which compositional deadlock checking approaches are applicable, are based on unrealistic assumptions, the number of component-based systems, to which compositional deadlock checking approaches are applicable, is limited significantly. Requiring fully bound components conflicts with the underlying reusability paradigm of CBSE, which often requires only parts of components or component compositions to be reused when defining new systems [1].

A component composition can for example be instantiated differently. That is, the set of components of the composition are configured slightly differently to be used in different contexts. The different configurations can for example differ w.r.t. the extent to which functionalities of the component composition are needed in a specific context but share a common topology. Usually different component implementations are used in different configurations. On the left hand side, Fig. 1 shows an excerpt from the CoCoME example [12], which describes the component-based design of a set of different trading systems. In the middle and on the right hand side Fig. 1 shows two possible configurations of the component composition. While *e1* includes only one store and therefore does not make use of the functionality to dispatch products between stores or report on an enterprise level, *e2* includes two stores, which are connected to the product dispatcher. In *e2* reporting is performed on store and enterprise level.

Consequently, it is desirable to reuse only parts of a component by connecting only the relevant functionalities. This, however, will likely lead to unbound or partially bound ports, which in turn will cause only a subset of some of the components' behavior to be executed. Currently, compositional deadlock checking approaches are not applicable to compositions with partially bound ports, as their restrictive assumptions are violated. From this the following problem can be concluded:

How can a component composition be checked for deadlock freedom, if instances of the composition can contain partial bindings?

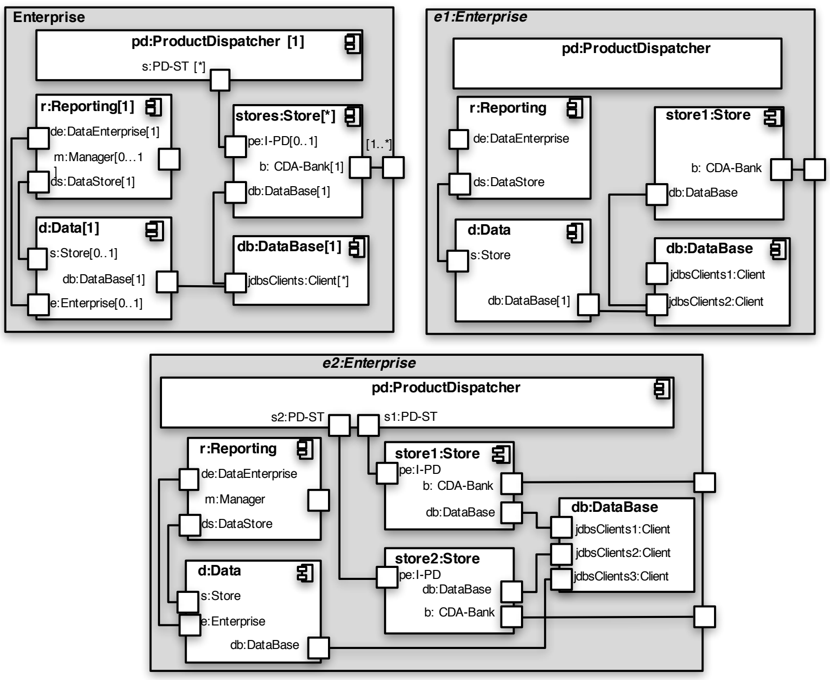

Fig. 1. Java/A-CoCoME enterprise composition specification (cf. [12]).

The remainder of this paper is structured as follows: Section 2 describes the solution idea in more detail. Section 3 presents related work. Section 4 describes the status of the current work and section 5 gives a short summary and outlook.

2 Solution Idea

An obvious solution to the described problem is to follow a brute force approach, in which all possible configurations with all possible combinations of partial bindings are instantiated and checked individually with traditional deadlock checking approaches. This solution has two problems:

1. *Combinatorial complexity:* The possible combinations of partial bindings in a composition result in a large space of configurations. Instantiating and verifying all possible configurations is not possible due to this combinatorial complexity.
2. *State explosion problem:* Each of the instantiated configurations has to be verified with traditional deadlock checking approaches due to the partial bindings. This implies computing the global composition behavior of each configuration and then performing the deadlock check on the global state behavior – both, which are subject to the state explosion problem.

So in conclusion, a brute force approach is not a viable option for verifying a component composition if configurations with partially bound components can be derived. Our solution idea intends to mitigate both problems and is therefore twofold.

2.1. Checking Desired Configurations

The first part of the solution idea intends to restrict the number of all possible configurations to a subset, which actually should be verified. Some functionalities might be core to a component composition, and thus it is unreasonable to exclude these functionalities in instantiations. In the eShop example depicted in figure 1, a store should for example always be connected to a database to allow goods to be purchased. Furthermore, there some functionalities can depend on each other in a composition. If the product-dispatching component (cf. Fig. 1) is for example not connected, related functionalities, like the reporting of statistics on product dispatches will not be relevant for the configuration, either. Thus, on a conceptual level there are dependencies between functionalities, which cause some combinations of partial bindings to be unreasonable. We propose to constrain the deadlock checking effort to a subset of configurations, which respects these dependencies and, is thus most likely to be used. All other configurations will be excluded from the deadlock check.

In software product line engineering (SPLE) variability modeling is a well-established technique to specify varying properties and qualities of a system (cf. [19]). We propose to explicitly model the subset of configurations to be verified by using modeling techniques from SPLE. A variability model restricts the variation of a product line by defining variation points, variants, and how the variants can be chosen and combined. We use the concepts of variability models to specify valid combinations of functionalities (and thus partial bindings) of compositions on a conceptual level. The main rational for constraining the configuration space of component compositions with variability models is that the number of options modelled in a variability model will be smaller than the theoretically possible space of possible configurations.

Limitations of this idea: A limitation of this solution is that it possible combinations of functionality to be anticipated. Thus, it still contradicts the reusability paradigm to some extend. However, we find that this is a reasonable trade-off, since i) it is a step towards relaxing the strict assumptions of compositional deadlock checking and ii) we are not focusing on anticipating all combinations by explicitly enumerating them but by explicitly stating dependencies and excluding unreasonable (and maybe erroneous) combinations.

2.2 Removing Superfluous Behavior

The second part of the solution idea intends to make partial bindings in a configuration explicit and remove the behavior, which is not executed because of those partial bindings. If this superfluous behavior is removed from the behavior specifications, compositional deadlock checking approaches are applicable to the reduced specifications. Using the variability model, it can be determined which functionalities have been excluded from an instance. Thus, it will be possible to identify partial bindings. In order to be able to derive behavior specifications for this configuration, which are reduced to the actual executable behavior, the effect of partial bindings and the combination of partial bindings onto the composition's behavior has to be analyzed.

We propose to identify and explicit specify these dependencies between partial bindings and the behavior of components in a behavior reduction model. Such a model can then be used to reduce the behavior specifications using existing model slicing techniques, where the partial bindings can be used as slicing criteria. Accordingly, such a model needs to provide means to relate the absence of functionality in a configuration (using the variability model and structural specification) to parts of the behavior specification. Furthermore, the notation needs to be compatible with existing slicing techniques. The resulting behavior specifications can then be used as input to existing compositional deadlock techniques. Thus, the state explosion problem is mitigated for the set of configurations, which is supposed to be verified.

Limitations of this idea: A risk to the proposed solution is that additional effort will be needed to generate the behavior reduction model and compute the reduced specifications. However, since the reduction model only has to be created once, we work under the hypothesis that the high number of derivable instances, in which the reduction model can be reused and for which the computation and analysis of the global state space can be avoided, will outweigh the extra effort. However, an experiment will be conducted in order to determine whether this hypothesis holds.

Fig. 2 shows an abstract sketch of the solution idea. It can be seen that the variability model will solely be related to the structural specification of the composition. The behavior reduction model is based on the behavior of the individual components and uses information from the structural specification as well as the variability model to enrich the behavior information.

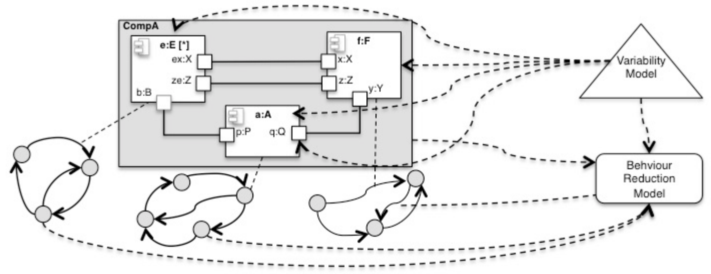

Fig. 2. Abstract solution sketch.

3 State-of-the-Art: Compositional Deadlock Checking

Deadlock-freedom is generally non-compositional: if components are deadlock free in isolation, they still might deadlock in a composition due to interactions making it challenging to find a sufficient condition to relate global and local deadlock-freedom. Existing compositional deadlock checking approaches can be differentiated into deductive and assume-guarantee reasoning approaches as described in the following.

3.1 Deductive Deadlock Checking

In summary, most deductive approaches focus on the pairwise behavioral equivalence between the behavior of connected components. Under the assumption that the components are deadlock free and that the interaction behavior is deadlock free, the analysis of the behavioral equivalence is a sufficient property to guarantee the local deadlock freedom of components and thus also the global deadlock freedom. Martens and Majster-Cederbaum's approach [10, 13, 14, 15] checks connected components for example pairwise for potential deadlocks. The reachability of those potential deadlocks is analyzed compositionally. The approach can only deal with tree like topologies. Although the approach theoretically allows for partial bindings, it is not e able to detect deadlocks that occur due to dependencies between several partial bindings.

In contrast, the approach presented by Aldini and Bernardo [2, 3, 7] also can also deal with cycles in arbitrary topologies. It is assumed that every topology can be reduced to an acyclic topology by reducing cycles into a new component. While the connections between the components are verified compositionally for the non-cyclic part, for the cyclic part they have to perform more extensive checks to assure that there are no cyclic dependencies between the components causing a deadlock. However, they avoid computing the "global" behavior of the components involved in the cycle by instead analyzing the communications paths within the cycle using a weak bisimulation equivalence relation. In contrast to Martens and Majster-Cederbaum their approach requires full bindings within a composition.

Choi and Kim's approach also assumes full binding of components in a composition [9]. It is based on controlled composition and abstraction. The approach provides a smaller behavior specification of a composite component by removing synchronized interaction in a parallel composition and removing the internal behavior of composite components via projection abstraction. The verification is performed compositionally for each component, whereas the dependencies towards the environment are considered by explicitly considering/providing drivers that can generate events and stubs that consume events. If each component behaves correctly under the assumption of a correct environment, the composition is be deadlock free.

Zeng and Miao [22] also propose an approach that is based on abstraction and compositional reasoning. As abstraction and compositional reasoning do not preserve deadlock freedom, the authors extend the notion of transitions in LTSs by classifying transitions into certain and uncertain transitions. The deadlock detection considers this differentiation and is performed using the CEGAR (counterexample guided abstraction refinement) framework. Even though the approach claims that it is compositional, their verification is not as it is based on the parallel composition of the abstractions.

Hennicker et al. [11, 12] propose an approach, which can compositionally assure deadlock freedom of a central component in a star topologies if at most one port of that component is behavioral restricted (partially bound). The approach is based on I/O transition systems and checks pairwise the behavior at connected ports fro deadlock freedom assuming that each component is deadlock free in isolation. The presented approach explicitly takes into account that components do not necessarily have to be fully bound, i.e. that expected behavior is not provided by the connected component. Hennicker et al. compute the parallel composition when more than one behavioral restriction is found. However, they only use the central component of the star topology and the ports at which behavioral restrictions occurred in the composition. Thus, they use the interaction behavior specification to detect behavioral restrictions and then directly check for deadlocks in the reduced global state space.

3.2 Assume-Guarantee Reasoning (AGR).

The idea of AGR [18] is to verify a component using an artificial environment that is based on the assumptions the component has about its environment. Using such an artificial environment reflecting the assumptions of a component, allows for an early verification of the component. Approaches focusing on how to use AGR for deadlock detection assume the parallel composition of two components to be deadlock free if the parallel composition of a component and its artificial environment is deadlock free and the other component "adheres" to the environment specification [8].

Chaki and Shina [8] for example propose a compositional deadlock detection algorithm that uses learning-based automated AGR. Their formalism is based on an automata-theoretic representation of failure traces. The deadlock detection is performed pairwise on failure automata and checks for inclusion in the specification language. They use an AGR rule, which uses the composition of a failure automaton with its assumption specification and then checks that a second automaton is included in the language of the assumptions.

Parizek and Plasil [16] propose several options to generate an artificial environment using AGR. The possibility to use only a subset of the functionality in the assumption specification enables the partial binding of components but all partial bindings have to be anticipated when specifying the assumptions.

Bensalem et al. [4, 5, 6] propose an approach for the compositional verification of components, which uses inductive reasoning as well as AGR. The approach is based on i) component invariants, which are an over-approximation of components' reachability sets and ii) interaction invariants, which are global constraints over the states of components involved in interactions. The overall algorithm sequentially computes a system invariant using the interaction and component invariants. If it cannot be shown that the desired invariant holds, a stronger invariant is computed or the calculation is stopped with inconclusive output. Like in the approach of Martens and Majster-Cederbaum, components can be partially reused. However, they do not discuss the problems, which could occur due to partial bindings nor is it obvious how the algorithm would be affected.

Generally, AGR approaches do not make any assumptions about the topology of a composition as long as the specified assumptions are satisfied. They allow the assumption specifications to be only a subset of the offered functionalities and thus allow partial bindings. However, any partial binding has to be anticipated and existing verification approaches do not explicitly check for deadlocks due to missing bindings.

4 Work in Progress

The research methodology we use, follows the idea of the design science research methodolgy and follows the six-step process model proposed by Peffers et al. [17]. The work currently in progress focuses on step 3 „Design and Development“, i.e. on developing an approach according to the two solution ideas. The work, which has to be performed as part of step 3 can be grouped according to the two core ideas of the solution idea as described in the following.

4.1 Part 1: Restricting the Combination Space

Step 1.1 Identification of appropiate means to restrict the configuration space. In order to restrict the configuration space of a component configuration which can be realized with partial bindings, an appropiate modeling notation has to be selected. As described in sect. 2, variability modeling as it is used in SPLE shows similarities with this objective and has therefore been selected as appropiate modeling means. Due to existing reasearch performed in our research group, the proposed approach is based on the Orthogonal Variability Model [19].

Step 1.2 Identification of variation in component compositions. In order to restrict the configuration space, it needs to be understood how configurations can vary and which elements of a composition are affected by unused functionality. Furthermore, the conceptual dependencies between functionalities need to be taken into account in this analysis. Different cases of variation have been identified in this step.

Step 1.3 Definition of artifact dependencies between the variability model and the composition. In SPLE, artifact dependencies describe the relation between development artifacts and the variability model and, therefore, determine, which development artifacts become part of a derived product [19]. The approach presented here also has to define artifact dependencies between the variability model and the component composition covering the cases identified in step 1.2. In SPLE a large number of approaches exist, which model variability in component compositions. These approaches distinguish between coarse grained variability (i.e. only the complete component can vary) and fine grained varaibility (i.e. single functionalities of a component can vary) in the component compositions. The approach proposed by this paper adopts a fine grained solution. The artifact dependencies have been specified formally extending the formal component model underlying the approach.

4.2 Part 2: Removing Superfluous Behavior

Step 2.1 Identification of the affect of partial bindings on behavior specifications. In order to remove superfluous behavior it has to be analysed how partial bindings can affect the behavior of a component. In this step, different cases have been identified, which differentiate between the partial binding of provided and required functionalities and which also consider the combination of several partial bindings and the implicit propagation of superfluous behavior onto other components.
Step 2.2 Identification of an appropiate modeling notation for the behavior reduction model. In order to allow superfluous behavior to be removed from behavior specfciations, an appropriate notation needs to be found for the behavior reduction model. Such a notation needs to meet the following requirements: i) it should be usable as input to existing model slicing techniques, ii) a relation between the variability model and the behavior reduction model needs to be defined in order to use the information of the variability model about the absence of functionalities and iii) the behavior reduction model also needs to use information from the structural specification about ports affected by the absence of functionalities.
Step 2.3 Definition of an algorithm to identify and remove superfluous behavior. Finally an algorithm needs to be defined, which allows an efficient identification of all cases defined in step 2.1 while considering the propagation between components and which also allows to remove superfluous behavior for a concrete composition.

5 Summary and Outlook

The paper motivates and describes a solution idea for allowing compositional deadlock checking approaches to be applied to compositions, which can be realized with partial bindings. The proposed approach is based on two core ideas: i) *restrict the number of configurations that can be derived and have to be verified* and *ii) remove behavior that is not executed due to partial bindings*.

The resulting behavior specifications can be used as input to existing compositional deadlock techniques. While first results have been achieved in defining the approach, the next step has to focus on defining the behavior reduction model and defining the algorithm for identifying and removing superfluous behavior before the applicability of the approach as well as the runtime experiments can be conducted.

The applicability of the approach will be illustrated with the CoCoME case study. Potential partial bindings will be analyzed for CoCoME and restricted with a variability model. An analysis of the number of all possible configurations in comparison to the number of restricted configurations will be conducted to demonstrate the usefulness of restricting the configuration space. In addition, the performance of the approach will be evaluated with experiments. These experiments will focus on analyzing the efficiency of the behavior reduction algorithm and compare the performance of compositional deadlock checking on reduced behavior specifications to the computation and analysis of the global state space for each possible configuration. A random subset of the derivable configurations of the CoCoME case study will be used as input to those experiments. The runtime will be compared to the runtime of traditional deadlock checking approaches, which compute the global system behavior using the same subset of configurations.

References

1. Adamek, J., Plasil, F.: Partial Bindings of Components - Any Harm? In: APSEC '04, IEEE CS, 632-639 (2004)
2. Aldini, A., Bernardo, M.: A General Approach to Deadlock Freedom Verification for Software Architectures. In: FME: Formal Methods, Springer, 2805, 658-677 (2003)
3. Aldini, A., Bernardo, M.: On the Usability of Process Algebra: An Architectural View. In: Theo. Comp. Sci., 335(2-3), 281–329 (2005)
4. Bensalem, S., Bozga, M., Nguyen, T. H., Sifakis, J.: Compositional verification for component-based systems and application. In: Software, IET, 4, 181-193 (2014)
5. Bensalem, S., Bozga, M., Legay, A., Nguyen, T.-H., Sifakis, J., Yan, R.: Incremental component-based construction and verification using invariants. In: FMCAD '10, 257-256 (2010)
6. Bensalem, S.; Griesmayer, A.; Legay, A.; Nguyen, T.-H., Peled, D.: Efficient deadlock detection for concurrent systems. In: MEMOCODE '11, 119-129 (2011)
7. Bernardo, M., Ciancarini, P., Donatiello, L.: Architecting Families of Software Systems with Process Algebras. In: ACM TOSEM, 11(4), 386–426 (2002)
8. Chaki, S., Sinha, N.: Assume-Guarantee Reasoning for Deadlock. In: FMCAD '06, 134-144 (2006)
9. Choi, Y., Kim, M.: Controlled composition and abstraction for bottom-up integration and verification of abstract components. In: Inf. Softw. Technol., Butterworth-Heinemann, 54, 119-136 (2012)
10. Gössler, G., Graf, S., Majster-Cederbaum, M., Martens, M., Sifakis, J.: An Approach to Modelling and Verification of Component Based Systems. In: SOFSEM '07, Springer, 4362, 295-308 (2007)
11. Hennicker, R., Janisch, S., Knapp, A.: On the Observable Behaviour of Composite Components. Electron. In: ENTCS, Elsevier, 260, 125-153. (2010)
12. Knapp, A., Janisch, S., Hennicker, R., Clark, A., Gilmore, S., Hacklinger, F., Baumeister, H., Wirsing, M.: Modelling the CoCoME with the Java/A Component Model. In: The Common Component Modeling Example, Springer, 5153, 207-237 (2008)
13. Lambertz, C., Majster-Cederbaum, M.: Efficient deadlock analysis of component-based software architectures. In: Science of Computer Programming, 78, 2488 – 2510 (2013)
14. Martens, M., Majster-Cederbaum, M.: Using Architectural Constraints for Deadlock-Freedom of Component Systems with Multiway Cooperation. In: TASE '09, IEEE CS, 225-232 (2009)
15. Martens, M., Majster-Cederbaum, M.: Deadlock-freedom in component systems with architectural constraints. In: Formal Methods in System Design, Springer, 41, 129-177 (2012)
16. Parizek, P., Plasil, F.: Assume-guarantee verification of software components in SOFA 2 framework. In: Software, IET, 4, 210-211 (2010)
17. Peffers, K., Tuunanen, T., Rothenberger, A. M., Chatterjee, S.: A design science research methodology for information systems research. In: JMIS, vol. 24, no. 3, 45-77, (2007)
18. Pnueli, A.: Logics and models of concurrent systems. In: Transition from global to modular temporal reasoning about programs. Springer, 123-144 (1985)
19. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering - Foundations, Principles, and Techniques. Springer (2005)
20. Semmelrock, N., Majster-Cederbaum, M.: Reachability in tree-like component systems is PSPACE- complete. In: ENTCS, Elsevier, 263, 197-210 (2010)
21. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley (2002)
22. Zeng, H., Miao, H.: Deadlock Detection for Parallel Composition of Components. In: Computer and Information Science, Springer, 317, 23-34 (2010)