Compositional Verification using Model Checking and Theorem Proving

Collaborative embedded systems form groups in which individual systems collaborate to achieve an overall goal. To this end, new systems may join a group and participating systems can leave the group. Classical techniques for the formal modeling and analysis of distributed systems, however, are mainly based on a static notion of systems and thus are often not well suited for the modeling and analysis of collaborative embedded systems. In this chapter, we propose an alternative approach that allows for the verification of dynamically evolving systems and we demonstrate it in terms of a running example: a simple version of an adaptable and flexible factory.

16.1 Introduction

Today more than ever, our daily life is determined by smart systems that are embedded into our environment. Modern systems even start collaborating with one another, making them collaborative embedded systems (CESs), which form collaborative system groups (CSGs). Due to the impact of such systems on modern society, verifying them has become an important task. However, their nature also imposes new challenges for verification.

Consider, for example, an adaptable and flexible factory as described in [Schlingloff 2018] and depicted in [Figure 16–1](#). Here, robots transport items between machines and together they form a CSG with the common goal of producing a complex item from simpler items. During the lifetime of the CSG, new individual CESs (robots or maybe even machines) may join the group while others may leave it.

Fig. 16–1: Smart production chain

Since traditional verification techniques usually focus on static system structures, they reach their limit when it comes to the verification of CSGs. Thus, in the following, we describe a novel approach to the verification of such systems that allows us to consider dynamically evolving groups of systems using a combination of automatic and semi-automatic verification techniques.

In this chapter, we first describe the approach in more detail. We then demonstrate it by applying it to the verification of a simple adaptable and flexible factory. We conclude with a brief summary, discussion of limitations, and outlook.

16.2 Approach

Figure 16-2 depicts an overview of our approach for the compositional verification of a CSG (represented as a group of individual CESs in the center). Verification of a set of overall system properties (represented by the list at the top right of the figure) proceeds in three steps: (i) We first identify suitable contracts for the individual CESs (represented by the filled boxes). (ii) We then verify the individual CESs against their contracts (left part of the figure). (iii) Finally, we combine the individual contracts with the description of the architecture to verify overall system properties (right-hand part of the figure).

Fig. 16-2: Hybrid verification approach

The approach is based on a formal system model that is based on FOCUS [Broy and Stolen 2012] and described in detail in [Marmsoler and Gleirscher 2016a], [Marmsoler and Gleirscher 2016b], and [Marmsoler 2019b]. To verify individual CESs against their contracts, we apply model checking [Clarke et al. 1986]. This allows us to change implementations of an individual CES and obtain fast feedback on whether the new implementation still satisfies the contracts of this CES.

Since we often do not know the exact number of CESs that participate in a CSG, we need to consider a possibly unlimited number of CESs. Thus, we apply interactive theorem proving [Nipkow et al. 2002] for the second step. The stability of results at the composition level justifies the additional effort that comes with interactive verification techniques compared to fully automatic techniques: as long as the single CESs satisfy their contracts, results at composition level remain valid.

To support a user in the development of specifications, the approach is implemented in terms of an Eclipse EMF-based modeling tool called FACTum Studio [Marmsoler and Gidey 2018], [Gidey et al.

2019]. The tool allows a user to develop specifications and proof sketches using a combination of graphical and textual modeling techniques. The specification can then be used to generate corresponding models and verification conditions for both the nuXmv [Cavada et al. 2014] model checker and the interactive theorem prover Isabelle/HOL [Nipkow et al. 2002].

To further support the development of interactive proofs, the approach comes with a framework to support the verification of dynamic architectures implemented in Isabelle [Marmsoler 2018b], [Marmsoler 2019c].

16.3 Example

To demonstrate the approach, we apply it to verify a simple property for our smart production use case.

16.3.1 Specification

We first need to specify the data types for the messages exchanged between the systems of our CSG. Figure 16-3 depicts a corresponding specification in terms of an abstract data type [Broy et al. 1984]: it specifies a data type *item* to represent the items produced in the system. For our example, we assume that items depend on one another in the sense that the production of a certain item may require another item. To this end, we specify a relationship $\leq_{it}$ between items such that $item1 \leq_{it} item2$ means that the production of an *item2* requires an *item1*. Note that the specification makes *items* an enumerable type, which allows us to use a successor function *succ* to obtain the successor of an item.

Fig. 16-3: Production items

As a next step, we have to specify the types of systems involved in our production chain. Figure 16-4 depicts a possible specification in terms of an architecture diagram [Marmsoler and Gidey 2019]: we specify two types of CES — machines and robots. Machines are parametrized by two items: one that represents the item a machine can produce, and one that represents the item the machine needs for the production. Thus, a system *Machine*(*item1*, *item2*) represents a machine that requires an *item1* to produce an *item2*. A robot, on the

other hand, is parametrized by a single item that represents the item it is able to carry. For example, a system $Robot(item1)$ is able to carry only items of type $item1$. In addition, the diagram requires that for every combination of items $it1, it2, it3$, where the production of $it3$ requires an $it2$ which in turn requires an $it1$, there is a machine $m1$ that can produce an item $it2$ when receiving an item $it1$ and a machine

Fig. 16-4: Architecture diagram for a smart production chain

$m2$ that requires an item $it2$ to produce an item $it3$, and a robot that can carry an item $it2$. Moreover, the diagram requires that the robot be connected to the machines via the correct ports, as depicted by the connections in the diagram.

Note that since we are using parameters here, the diagram actually specifies a production sequence of arbitrary length depending on the concrete items provided. Moreover, the specification allows individual CESs to leave and join the production chain as long as the architectural property is satisfied. For example, a robot may leave the CSG if there is another robot that can take over its responsibilities.

After specifying the architecture, we can specify the behavior of individual types of systems. [Figure 16-5](#) depicts a simplified specification of a possible machine implementation in terms of a state machine: a machine waits for a source item before starting the production and delivering the item. In addition to the implementation, we must also specify contracts for the system using linear temporal logic [Manna and Pnueli 1992]. The contract specified for a machine in [Figure 16-5](#), for example, states that whenever a machine obtains

the required input item, it will eventually produce the desired output item.

Fig. 16-5: Specification of a machine

Note that we use the machine's parameters *item1* and *item2* in formulating the contract.

Similarly, we have to specify the implementation of a robot, which is depicted in Figure 16-6: a robot collects an item, moves around, and finally drops the item when it reaches the correct position. Again, we have omitted details about the guards for the transitions for the sake of readability. And again, we also formulated a possible contract for a robot at the bottom of the diagram stating that a robot will always deliver a collected item.

Fig. 16-6: Specification of a robot

16.3.2 Verification

Let us assume, for the purpose of our example, that we want to verify that the CSG can produce the final production item of a chain of arbitrary length, given that it is provided with the first item required in the chain. For example, if we are given a chain of items $item1 \leq_{it} item2 \leq_{it} \dots \leq_{it} itemN$, then our group should be able to collaboratively produce item $itemN$ when it receives a corresponding $item1$.

As shown in [Figure 16-2](#), verifying a specification of a CSG consists of two parts: first, we apply model checking to verify that a single component indeed satisfies its contracts. If we use FACTum Studio to model our system, we could then simply generate a model and corresponding verification conditions for the nuXmv model checker from the specification to automatically perform the verification.

Next, we have to combine the individual contracts to show that the overall system works correctly. To do so, we first show a smaller result that states that for every machine-robot-machine combination, when the first machine receives the correct input item, the second machine provides the correct output. Note that this involves combining three different contracts: the two contracts that ensure that the two machines function correctly, and another contract that ensures that the robot functions correctly. We can sketch this proof using an architecture proof modeling language (APML) [Marmsoler and Blakqori 2019], a notation similar to a sequence chart for sketching composition proofs. A possible APML proof sketch is shown in [Figure 16-7](#): it first states the property in linear temporal logic at the top and then provides a proof sketch in the form of a sequence diagram. The proof sketch describes how the different contracts need to be combined to discharge the overall proof obligation.

Note the reference to the corresponding contracts: production, delivery, production.

Again, if we use FACTum Studio for the specification of the APML proof sketch, then we can automatically generate a corresponding proof for the interactive theorem prover Isabelle to check the soundness of the proof sketch.

Fig. 16-7: APML proof for smart production

The result we just proved shows the correctness of one segment of our production chain. Now, to show the correctness of the complete chain, we have to repeat our argument for every segment of the chain. We can do this using a technique called *well-founded induction* [Winskel 1993]. The corresponding sketch is shown in Figure 16-8. This concludes the proof and therefore the verification of our production chain.

Fig. 16-8: Well-founded induction over production chain

16.4 Conclusion

In this chapter, we described an approach for verifying CSGs based on a combination of automatic and semi-automatic verification techniques and we demonstrated our approach in terms of a simple example. As shown by the example, the approach allows verification of CSGs that consist of an arbitrary number of individual CESs. Thus, it complements traditional verification approaches that usually assume a static structure with a fixed number of systems involved.

In addition to the example described in this chapter, the approach has been successfully applied to other domains as well, such as train control systems [Marmsoler and Blakqori 2019], architectural design patterns [Marmsoler 2018a], and even blockchain [Marmsoler 2019a]. While this showed the general feasibility of the approach, it also revealed some limitations: one weakness concerns the expressive power of our contracts. As of now, contracts are limited to a restricted form of linear temporal logic and many interesting properties cannot be expressed. Thus, future works should investigate alternative notions of contracts to increase expressiveness. Another weakness concerns the generation of Isabelle proofs from APML proof sketches. Sometimes, the proofs generated do not contain all the necessary details required by Isabelle to confirm the proof and some manual additions may be necessary. Thus, future work should also investigate possibilities to generate more complete proofs to minimize interactions with the interactive theorem prover. Finally, our system model assumes the existence of a global time to synchronize different components. While this assumption is suitable for some scenarios, there might be other scenarios where it might not hold. Thus, future work should investigate possibilities to weaken this assumption.

16.5 Literature

- [Broy et al. 1984] M. Broy, M. Wirsing, C. Pair: A Systematic Study of Models of Abstract Data Types. In: *Theoretical Computer Science*, vol. 33, 1984, pp. 139-174.
- [Broy and Stolen 2012] M. Broy, K. Stolen: *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*, Springer Science & Business Media, 2012.
- [Cavada et al. 2014] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, S. Tonetta: The nuXmv Symbolic Model Checker. In: Biere A., Bloem R. (eds) *Computer Aided Verification*. CAV 2014.
- [Clarke et al. 1986] E. M. Clarke, E. A. Emerson, A. P. Sistla: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. In: *ACM Trans. Program. Lang. Syst.*, vol. 8, 4 1986, pp. 244-263.

- [Gidey et al. 2019] H. K. Gidey, A. Collins, D. Marmsoler: Modeling and Verifying Dynamic Architectures with FACTum Studio. In: Arbab F., Jongmans SS. (eds) Formal Aspects of Component Software. FACS 2019.
- [Manna and Pnueli 1992] Z. Manna, A. Pnueli: The Temporal Logic of Reactive and Concurrent Systems, Springer New York, 1992.
- [Marmsoler and Gleirscher 2016a] D. Marmsoler, M. Gleirscher: Specifying Properties of Dynamic Architectures Using Configuration Traces. In: International Colloquium on Theoretical Aspects of Computing, Springer, 2016, pp. 235–254.
- [Marmsoler and Gleirscher 2016b] D. Marmsoler, M. Gleirscher: On Activation, Connection, and Behavior in Dynamic Architectures. In: Scientific Annals of Computer Science, vol. 26, 2016, pp. 187–248.
- [Marmsoler 2018a] D. Marmsoler: Hierarchical Specification and Verification of Architecture Design Patterns. In: Fundamental Approaches to Software Engineering - 21th International Conference, FASE 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14–20, 2018, Proceedings, 2018.
- [Marmsoler and Gidey 2018] D. Marmsoler, H. K. Gidey: FACTUM Studio: A Tool for the Axiomatic Specification and Verification of Architectural Design Patterns. In: Formal Aspects of Component Software - FACS 2018 - 15th International Conference, Proceedings, 2018.
- [Marmsoler 2018b] D. Marmsoler: A Framework for Interactive Verification of Architectural Design Patterns in Isabelle/HOL. In: The 20th International Conference on Formal Engineering Methods, ICFEM 2018, Proceedings, 2018.
- [Marmsoler and Blakqori 2019] D. Marmsoler, G. Blakqori: APML: An Architecture Proof Modeling Language. In: Formal Methods – The Next 30 Years, Cham, 2019.
- [Marmsoler 2019a] D. Marmsoler: Towards Verified Blockchain Architectures: A Case Study on Interactive Architecture Verification. In: Formal Techniques for Distributed Objects, Components, and Systems, Cham, 2019.
- [Marmsoler 2019b] D. Marmsoler: A Denotational Semantics for Dynamic Architectures. In: 2019 International Symposium on Theoretical Aspects of Software Engineering (TASE), 2019.
- [Marmsoler 2019c] D. Marmsoler: A Calculus for Dynamic Architectures. In: Science of Computer Programming, vol. 182, 2019, pp. 1–41.
- [Marmsoler and Gidey 2019] D. Marmsoler, H. K. Gidey: Interactive Verification of Architectural Design Patterns in FACTum. In: Formal Aspects of Computing, vol 31, 2019, pp. 541–610.
- [Nipkow et al. 2002] T. Nipkow, L. C. Paulson, M. Wenzel: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, vol. 2283, Springer Science & Business Media, 2020.
- [Schlingloff 2018] B. Schlingloff: Specification and Verification of Collaborative Transport Robots, in 4th International Workshop on Emerging Ideas and Trends in the Engineering of Cyber-Physical Systems (EITEC), 2018.
- [Winskel 1993] Glynn Winskel: The Formal Semantics of Programming Languages: An Introduction. MIT Press, Cambridge, MA, USA, 1993.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.