Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 180 (2021) 502–506

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Industry 4.0 and Smart Manufacturing

Evaluating the alignment of sequence diagrams with system behavior

Atif Mashkoor[a,b], Alexander Egyed[b]

[a] *Software Competence Center Hagenberg GmbH, Hagenberg, Austria*
[b] *Johannes Kepler University, Linz, Austria*

Abstract

In model-driven engineering, sequence diagrams are commonly used to describe a system's expected behavior in different scenarios. Indeed, the information flow described in sequence diagrams should actually take place during a real execution of the system in order to ensure its safety, security and correctness. If it does not, this may lead to serious consequences. In this short paper, we present a novel generic approach for addressing this issue by observing the live execution of a system and checking whether the exhibited information flow correctly follows what has been specified in sequence diagrams.

Keywords: Sequence diagrams; run-time behavior; correctness;

1. Introduction

Sequence diagrams are a common way in model-driven engineering to specify desired information flows in different usage scenarios of a system [12]. Sequence diagrams provide elements that are similar to control flow related constructs in programming languages (e.g., loops or alternative paths). Yet, implementation details, such as message parameters or arguments, are often omitted for two reasons. First, it makes the sequence diagrams easier to comprehend for stakeholders. Second, it is desired that the actual implementation is decided by the responsible developer and not by a system architect or a designer.

In practice, sequence diagrams remain a depiction of what components' interactions should look like, and source code is still written manually by engineers rather than being generated automatically using code generators. Yet, it is, of course, critical that the information flow modeled in sequence diagrams is also exhibited by the system when it is actually executed. If the run-time behavior of the system differs from what is defined in sequence diagrams, this

E-mail address: firstname.lastname@jku.at

10.1016/j.procs.2021.01.267

clearly indicates that either the implementation is incorrect (i.e., the engineers manually implementing the sequence diagrams made a mistake) or the specification of the desired behavior is incorrect (i.e., the designer did not fully understand what the system should actually do in the modeled scenario). The former is subject to software testing and the latter is subject to requirements validation [9]. Anyway, both possibilities need to be clarified and eliminated as they may lead to severe safety and security issues [2, 8, 10].

Currently, such misalignments are detected only after the release or deployment of a system. As it has been discussed frequently in literature, such as by Boehm et al. [3], detecting such errors becomes harder, and with that more expensive, to fix the longer they remain undetected. Therefore, it is crucial to detect misalignments between sequence diagrams and run-time behavior of a system as early in the development process as possible.

In this short paper, we address this issue by presenting a novel generic approach that evaluates the alignment of sequence diagrams and system behavior. The approach relies on the actual execution of the system. In particular, it observes the system during its execution and checks live whether the observed behavior reflects the system's desired behavior as defined in sequence diagrams. In order to be able to check this efficiently, our approach relies on non-deterministic finite automata (NFA) that are generated from sequence diagrams automatically. The approach is applicable to any implementation language and execution platform for as long as the execution of the system can be observed to create a stream of messages sent between individual components. Therefore, the approach can be used already in early stages of development.

To validate our approach and to demonstrate its feasibility, we implemented a prototype tool, which takes as input UML sequence diagrams. For the information about the actual execution of the system, this prototype uses a state machine simulator to generate events. We employed the prototype to conduct case studies with four different systems for which both sequence diagrams and state machines were available. The validation results indicate that our proposed approach identifies misalignments.

The paper is organized as following: Sec. 2 presents the approach for evaluating the alignment of sequence diagrams and the run-time behavior of a system. Sec. 3 presents the validation of the proposed approach. Sec. 4 presents a quick glimpse of the related work. The paper is finally concluded in Sec. 5.

2. Approach

We now present an overview of our approach for evaluating the alignment of sequence diagrams and the run-time behavior of a system, which is depicted in Fig. 1. The proposed approach relies on the execution of the system and the observation of its behavior through a monitor component. Specifically, any information flow that is observed during the system's execution by the monitor is captured in a message stream (also called trace). Our approach automatically checks whether this trace is allowed by the defined sequence diagrams. This checking is done live during the system's execution (or afterwards, if this is desired) by a dedicated checker. To conduct the checking in an efficient way, the checker relies on NFA that are constructed automatically from the sequence diagrams via a sequence diagram compiler component.

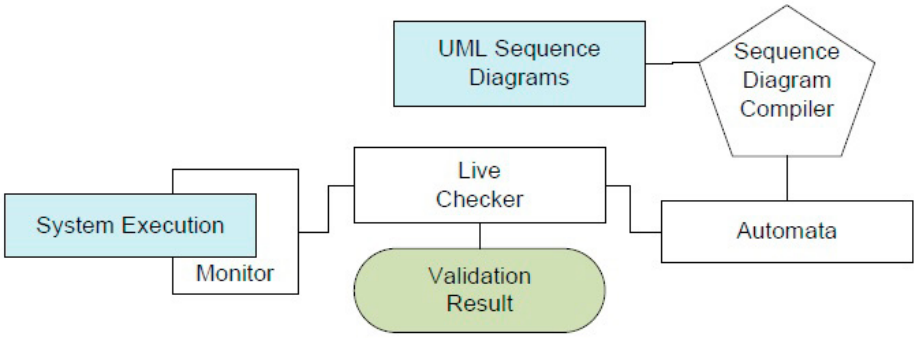

Fig. 1. Architectural overview of the approach

2.1. Input

In order to apply our approach, the following two requirements must be fulfilled: 1) usage scenarios of the system are defined as UML sequence diagrams, and 2) the system is executed in an observable environment that provides basic data about information flows that take place during execution. The input artifacts for our approach are highlighted with a light blue background in Fig. 1.

2.2. Output

The output of our approach, which is highlighted with a light green background in Fig. 1, is primarily information about whether the system's execution was violating any specifications of the sequence diagrams. However, additional information is also available. For example, which usage scenarios (i.e., which sequence diagrams) have been executed correctly or incorrectly by the system or which specific order of events lead to these results.

3. Validation

To validate our approach, we developed a prototype implementation and applied the approach to four different systems. Moreover, we also talk about the correctness of our approach.

3.1. Prototype Implementation

To demonstrate the feasibility of our approach, we built a prototype implementation that uses as input standard UML sequence diagrams and generates events by observing a state diagram simulator (SDS) [4]. The tool is implemented as a plug-in for the IBM Rational Software Architect (RSA) [6] and is available online[1]. Using the RSA standard features, users can define models using class diagrams, sequence diagrams, and state machines. Using the SDS, users can then simulate an execution of the system. The plug-in implementing our approach observes such executions live and tests them against the defined state machines. A graphical user interface provides live information about which scenarios have been executed validly or invalidly.

3.2. Case Studies

The prototype implementation discussed above was used with models of four different systems. The models had on average five state machines and two sequence diagrams with, on average, 4.5 lifelines per sequence diagram. For more information about the used models, we refer to [11]. Our tool was able to generate NFAs for all sequence diagrams. Moreover, it recognized all sequence diagrams when the state machines behaved accordingly during the simulation.

3.3. Correctness

As far as formal correctness is concerned, it has been proven that the original Thompson construction [1] – transformation of a regular expression into an equivalent NFA – is correct. Since our construction algorithm follows the Thompson algorithm for message occurrences, sequences, alternatives, and loop combined fragments, we must assume that these parts of the generated NFA are correct as well. Furthermore, all additionally added construction rules mainly maintain properties of the original algorithm such as every NFA has one start state (without incoming transitions) and one accepting state (without outgoing transitions). Therefore, we can assume with confidence that the NFA construction of our approach is correct.

[1] https://doi.org/10.6084/m9.figshare.13084232.v1

4. Related Work

Some existing approaches also try to integrate sequence diagrams by using them for synthesis. In model-driven engineering, synthesis is used to generate code from sequence diagrams and to ensure that the system cannot behave in an unintended way during its execution. For this purpose, many researchers proposed formal semantics for sequence diagrams such as Grosu et al. [5]. In contrast to code synthesis, other approaches synthesize state charts, and thus behavior descriptions, out of sequence diagrams [7], which is more closely related to our goal. However, the resulting state charts are hard to read and describe the system's behavior incompletely, due to the scenario nature of sequence diagrams. Therefore, a semi-automatic generation of state charts has been proposed [13]. The need for a human input in the synthesis of state charts out of sequence diagrams proves that sequence diagrams were not designed for creating a whole model. In particular, the need for manual input makes it possible that the resulting system exhibits behavior that was not intended in the sequence diagrams. This means that the checking our approach performs is still necessary.

5. Conclusion

In this paper, we have presented an approach that evaluates the alignment of sequence diagrams and the actual behavior of a system live during its execution. The implementation for sequence diagrams and a state machine simulator demonstrates the feasibility of the approach. Additionally, we have shown the correctness of the approach. The approach is validated by the empirical data we observed during the testing of four case studies, for which we identified whether defined scenarios were executed validly.

Acknowledgment

The research reported in this paper has been partly funded by the Austrian Science Fund (FWF) in the framework of the IVOIRE project (I 4744-N), the LIT Secure and Correct System Lab, and the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry for Digital and Economic Affairs, and the Province of Upper Austria in the frame of the COMET - Competence Centers for Excellent Technologies - program managed by FFG.

References

[1] Aho, A.V., Sethi, R., Ullman, J.D., 1986. Compilers: Principles, Techniques, and Tools. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[2] Biró, M., Mashkoor, A., Sametinger, J., Seker, R., 2018. Software safety and security risk mitigation in cyber-physical systems. IEEE Softw. 35, 24–29. URL: https://doi.org/10.1109/MS.2017.4541050, doi:10.1109/MS.2017.4541050.

[3] Boehm, B., Koolmanojwong, S., Lane, J.A., Turner, R., 2012. Principles for successful systems engineering. Procedia Computer Science 8, 297 – 302. URL: http://www.sciencedirect.com/science/article/pii/S1877050912000646, doi:https://doi.org/10.1016/j.procs.2012.01.063. conference on Systems Engineering Research.

[4] Egyed, A., Wile, D., 2001. Statechart simulator for modeling architectural dynamics, in: Proceedings Working IEEE/IFIP Conference on Software Architecture, pp. 87–96. doi:10.1109/WICSA.2001.948413.

[5] Grosu, R., Smolka, S.A., 2005. Safety-liveness semantics for uml 2.0 sequence diagrams, in: Fifth International Conference on Application of Concurrency to System Design (ACSD'05), pp. 6–14. doi:10.1109/ACSD.2005.31.

[6] Leroux, D., Nally, M., Hussey, K., 2006. Rational software architect: A tool for domain-specific modeling. IBM Systems Journal 45, 555–568. doi:10.1147/sj.453.0555.

[7] Liang, H., Dingel, J., Diskin, Z., 2006. A comparative survey of scenario-based to state-based model synthesis approaches, in: Proceedings of the 2006 International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools, ACM, New York, NY, USA. pp. 5–12. URL: http://doi.acm.org/10.1145/1138953.1138956, doi:10.1145/1138953.1138956.

[8] Mashkoor, A., Biró, M., Messnarz, R., Palacios, R.C., 2018. Selected functional safety and cybersecurity concerns in system, software, and service process improvement and innovation. J. Softw. Evol. Process. 30. URL: https://doi.org/10.1002/smr.1955, doi:10.1002/smr.1955.

[9] Mashkoor, A., Matoussi, A., 2010. Towards validation of requirements models, in: The Second International Conference on ASMs, Alloy, B and Z (ABZ 2010), Springer, Orford (Québec), Canada. p. 404. doi:10.1007/978-3-642-11811-1_38.

[10] Mashkoor, A., Sametinger, J., Biró, M., Egyed, A., 2020. Security- and safety-critical cyber-physical systems. J. Softw. Evol. Process. 32. URL: https://doi.org/10.1002/smr.2239, doi:10.1002/smr.2239.

[11] Mitterer, P., 2014. Using interactions to validate executing state machines, Master thesis. Johannes Kepler University Linz. URL: http://140.78.115.16/bibPublications/MSc%20Theses/2014%20Philipp%20Mitterer/Mitterer2014%20-%20Using%20Interactions%20to%20Validate%20Executing%20State%20Machines.pdf.

[12] Schmidt, D.C., 2006. Model-driven engineering. COMPUTER-IEEE COMPUTER SOCIETY- 39, 25.

[13] Whittle, J., Schumann, J., 2000. Generating statechart designs from scenarios, in: Proceedings of the 22nd International Conference on Software Engineering, ACM, New York, NY, USA. pp. 314–323. URL: http://doi.acm.org/10.1145/337180.337217, doi:10.1145/337180.337217.