Automated Scenario-Based Integration Testing of Distributed Systems

Bruno Lima
INESC TEC and Faculty of Engineering, University of Porto
Porto, Portugal
bruno.lima@fe.up.pt

ABSTRACT

In a growing number of domains, the provisioning of end-to-end services to the users depends on the proper interoperation of multiple systems, forming a new distributed system, often subject to timing constraints. To ensure interoperability and integrity, it is important to conduct integration tests that verify the interactions with the environment and between the system components in key scenarios. To tackle test automation challenges, we propose algorithms for decentralized conformance checking and test input generation, and for checking and enforcing the conditions (local observability and controllability) that allow decentralized test execution.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**;

KEYWORDS

Scenario-based Testing, Integration Testing, Distributed Systems

ACM Reference Format:
Bruno Lima. 2018. Automated Scenario-Based Integration Testing of Distributed Systems. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3236024.3275431

1 PROBLEM AND MOTIVATION

To ensure interoperability and the integrity of distributed systems or systems of systems, it is important to conduct integration tests that verify the interactions with the environment and between the system components in key scenarios. Integration test scenarios for that purpose may be conveniently specified by means of UML Sequence Diagrams [16][5] (SDs), because they are a industry standard well suited for describing and visualizing the interactions that occur between the components and actors of a distributed system under test (SUT), possibly including time constraints. As a motivating example, Figure 1 models a test scenario for a fall detection system with three participants.

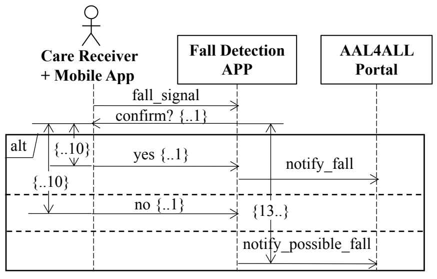

Figure 1: Example integration testing scenario.

Our goal is to develop algorithms for decentralized conformance checking and test input generation, and for checking and enforcing the conditions (local observability and controllability) that allow decentralized test execution, in the context of scenario-based testing of time-constrained distributed systems.

2 BACKGROUND AND RELATED WORK

2.1 Model-Based Testing

MBT approaches have the potential to increase the effectiveness and efficiency of the test process, by means of automatic test case generation from behavioral models of the SUT [19], but suffer from several limitations [2], such as the test case explosion problem and the lack of support for the integration testing of distributed systems.

MBT can be done *offline* or *online*, meaning that test generation and execution are performed separately or together, respectively [17][14]. We focus our work on the *online* testing of distributed systems, because it is adequate to cope with non-determinism [20].

MBT approaches use a high variety of models, from state-based to scenario-based ones [4]. We use *scenario-based* models, because they are well suited for describing the interactions between the components and actors of a distributed system [12] in key scenarios, and help reducing the test case explosion problem.

2.2 Test Architectures for Distributed Systems

Three architectures for testing distributed systems have been proposed in the past, with different fault detection capabilities [8]: purely distributed, with independent *local testers* communicating with the SUT components [18]; purely centralized, with a single *central tester*; hybrid [7], combining the previous approaches to improve fault detection. Given its advantages, we base our work on the *hybrid test architecture* [7], with some extensions to also check the interactions between the SUT components and further minimize the communication overhead.

2.3 Observability and Controllability in Distributed Testing

Observability and controllability are a common issue in distributed system testing, addressed by some works in the literature. E.g., an algorithm was proposed in [6] for introducing coordination messages to overcome controllability problems in system testing from an input output transition system, but they only consider one event per message, whilst integration testing requires the consideration of distinct emission and reception events. In [15], solutions are proposed to overcome controllability problems related with race conditions in scenarios described through MSCs or UML SDs, but only basic scenarios are addressed, without control flow variants.

3 APPROACH AND CONTRIBUTIONS

3.1 Overall Approach and Architecture

In previous work [12] we proposed an approach and toolset architecture for automating the testing of end-to-end services in distributed and heterogeneous systems, comprising a visual modeling environment and a distributed test input generation and conformance checking engine (see Figure 2). In that approach, the only manual activity required from the tester is the description of the participants and behavior of the services under test with UML SDs (together with mapping information between the model and the implementation), which are automatically translated to a formal notation for efficient test input generation and conformance checking at runtime.

In order to be able to check not only the interactions with the environment but also the interactions between the system components, a *local tester* is deployed close to each system component, and a *central tester* coordinates the local testers. The central and local testers are SUT independent, but local testers may need a platform adapter.

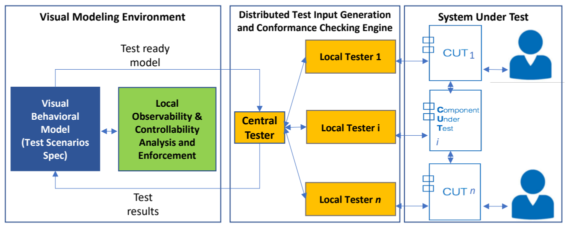
Visual Modeling Environment
Test ready model
Visual Behavioral Model (Test Scenarios Spec)
Local Observability & Controllability Analysis and Enforcement
Test results
Distributed Test Input Generation and Conformance Checking Engine
Central Tester
Local Tester 1
Local Tester i
Local Tester n
System Under Test
CUT 1
Component Under Test i
CUT n

Figure 2: Model-based integration testing architecture.

3.2 Local Observability and Controllability Analysis and Enforcement

In such a test architecture, it is important to minimize the communication overhead during test execution, namely in the presence of time constraints. It is also important to detect errors as early as possible and closely as possible to the offending component, to facilitate fault detection and localization. In ideal scenarios, after the central tester sets up and initiates the local testers for a given test scenario (described by a SD), no communication between the test components should occur during test execution, and the central tester would only need to receive a verdict from each local tester at the end of successful test execution or as soon as an error is detected. We say that a given scenario is *locally observable* and *locally controllable* if observed events can be checked locally and inputs to inject can be decided locally by the local testers, without the needed for exchanging coordination messages between the test components during test execution.

Since practical scenarios may not exhibit those properties, we are investigating procedures to check their local observability and controllability, and determine minimal sets of additional coordination messages and timing constraints needed to enforce those properties. E.g., the test scenario shown in Figure 1 is locally controllable, but would not be locally controllable without the time constraints. The scenario is not locally observable, because the transmission time constraints cannot be checked locally by the lifelines.

In previous work [13], we formalized procedures for local observability and controllability checking in the absence of time constraints, using the VDM++ formal specification language [3][10] supported by the Overture tool [9]. We also provided examples of local observability and controllability problems caused by race conditions, non-local choices, and optional messages without corresponding acknowledgment messages. We are currently extending those procedures for time-constrained scenarios and developing the enforcement algorithms.

3.3 Distributed Conformance Checking and Test Input Generation

In previous work we developed procedures to incrementally check the conformance of observed execution traces against a test specification set by a UML SD, in the context of integration testing of distributed system, firstly without time constraints [13] and subsequently with time constraints and no global clock [11]. Due to the distributed observation of execution events and to the impossibility to ensure clock synchronization in a distributed system, the test verdict reached may be inconclusive in some cases.

In order to cope with non-determinism, we follow an online (or adaptive) MBT approach, in which test inputs (what and when) are decided based on the events observed so far at each lifeline. Following an online MBT approach in a distributed setting is particularly challenging. In future work, we will develop distributed algorithms for online distributed test input generation.

4 CONCLUSIONS

Given the growing importance of distributed systems testing, we are developing novel algorithms for decentralized conformance checking and test input generation and for local observability and controllability analysis and enforcement in scenario-based integration testing of time-constrained distributed systems.

We intend to experimentally evaluate our approach in real world case studies in the IoT e-health domain, regarding the communication overhead and the fault detection and localization capabilities, as compared to other MBT automation approaches for distributed systems described in the literature (e.g., [1]).

ACKNOWLEDGMENT

This work was financed by the Portuguese Foundation for Science and Technology (FCT), under research grant SFRH/BD/115358/2016.

REFERENCES

[1] Aivo Anier, Jüri Vain, and Leonidas Tsiopoulos. 2017. DTRON: a tool for distributed model-based testing of time critical applications. *Proceedings of the Estonian Academy of Sciences* 66, 1 (2017).

[2] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. 2007. A Survey on Model-based Testing Approaches: A Systematic Review. In *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22Nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007 (WEASELTech '07)*. ACM, New York, NY, USA, 31–36. https://doi.org/10.1145/1353673.1353681

[3] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, and Marcel Verhoef. 2005. *Validated Designs For Object-oriented Systems*. Springer-Verlag TELOS, Santa Clara, CA, USA.

[4] Wolfgang Grieskamp. 2006. *Formal Approaches to Software Testing and Runtime Verification: First Combined International Workshops, FATES 2006 and RV 2006, Seattle, WA, USA, August 15-16, 2006, Revised Selected Papers*. Springer Berlin Heidelberg, Berlin, Heidelberg, Chapter Multi-paradigmatic Model-Based Testing, 1–19. https://doi.org/10.1007/11940197_1

[5] Hans-Gerhard Gross. 2005. *Component-Based Software Testing with UML*. Springer Berlin Heidelberg.

[6] Robert M. Hierons. 2012. Overcoming controllability problems in distributed testing from an input output transition system. *Distributed Computing* 25, 1 (2012), 63–81. https://doi.org/10.1007/s00446-011-0153-5

[7] Robert M. Hierons. 2014. Combining Centralised and Distributed Testing. *ACM Trans. Softw. Eng. Methodol.* 24, 1, Article 5 (Oct. 2014), 29 pages. https://doi.org/10.1145/2661296

[8] Robert M. Hierons, Mercedes G. Merayo, and Manuel Núñez. 2011. Scenarios-based testing of systems with distributed ports. *Software: Practice and Experience* 41, 10 (2011), 999–1026. https://doi.org/10.1002/spe.1062

[9] Peter Gorm Larsen, Nick Battle, Miguel Ferreira, John Fitzgerald, Kenneth Lausdahl, and Marcel Verhoef. 2010. The overture initiative integrating tools for VDM. *ACM SIGSOFT Software Engineering Notes* 35, 1 (2010), 1–6.

[10] Peter G. Larsen, Kenneth Lausdahl, Nick Battle, John Fitzgeral, Sune Wolff, Shin Sahra, Marcel Verhoef, Peter Tran-JÃÿrgensen, Tomohiro Oda, and Paul Chisholm. 2016. *VDM-10 Language Manual*. Technical Report.

[11] Bruno Lima and João Faria. 2017. Conformance Checking in Integration Testing of Time-constrained Distributed Systems based on UML Sequence Diagrams. In *Proceedings of the 12th International Conference on Software Technologies - Volume 1: ICSOFT,*. INSTICC, SciTePress, 459–466. https://doi.org/10.5220/0006474004590466

[12] Bruno Lima and João Pascoal Faria. 2016. *Software Technologies: 10th International Joint Conference, ICSOFT 2015, Colmar, France, July 20-22, 2015, Revised Selected Papers*. Springer International Publishing, Cham, Chapter Automated Testing of Distributed and Heterogeneous Systems Based on UML Sequence Diagrams, 380–396. https://doi.org/10.1007/978-3-319-30142-6_21

[13] B. M. C. Lima and J. C. P. Faria. 2017. Towards Decentralized Conformance Checking in Model-Based Testing of Distributed Systems. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 356–365. https://doi.org/10.1109/ICSTW.2017.64

[14] Marius Mikucionis, Kim G Larsen, and Brian Nielsen. 2004. T-uppaal: Online model-based testing of real-time systems. In *Automated Software Engineering, 2004. Proceedings. 19th International Conference on*. IEEE, 396–397.

[15] B. Mitchell. 2005. Resolving race conditions in asynchronous partial order scenarios. *IEEE Transactions on Software Engineering* 31, 9 (Sept 2005), 767–784. https://doi.org/10.1109/TSE.2005.104

[16] OMG. 2015. *OMG Unified Modeling Language TM (OMG UML) Version 2.5*. Technical Report. Object Management Group.

[17] Stephan Schulz, Jukka Honkola, and Antti Huima. 2007. Towards model-based testing with architecture models. In *Engineering of Computer-Based Systems, 2007. ECBS'07. 14th Annual IEEE International Conference and Workshops on the*. IEEE, 495–502.

[18] Andreas Ulrich and Hartmut König. 1999. Architectures for Testing Distributed Systems. In *Testing of Communicating Systems*, Gyula Csopaki, Sarolta Dibuz, and Katalin Tarnay (Eds.). IFIP - The International Federation for Information Processing, Vol. 21. Springer US, 93–108. https://doi.org/10.1007/978-0-387-35567-2_7

[19] Mark Utting and Bruno Legeard. 2007. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[20] Mark Utting, Alexander Pretschner, and Bruno Legeard. 2012. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability* 22, 5 (2012), 297–312. https://doi.org/10.1002/stvr.456