Towards Testing Self-organizing, Adaptive Systems

Benedikt Eberhardinger, Hella Seebach, Alexander Knapp, Wolfgang Reif

Towards Testing Self-Organizing, Adaptive Systems

Benedikt Eberhardinger, Hella Seebach, Alexander Knapp, and Wolfgang Reif

Institute for Software & Software Engineering, University of Augsburg, Germany
{benedikt.eberhardinger, seebach, knapp, reif}@informatik.uni-augsburg.de

Abstract. The characteristics of self-adaptive, self-organizing systems lead to a significant higher flexibility and robustness against a changing environment. This flexibility makes it hard to test these systems adequately. To assure their quality, however, it is inevitable to do so. We introduce a new approach for systematically testing these self-* systems based on a feedback control-oriented system architecture called Corridor Enforcing Infrastructure (CEI). Thus, it is possible to examine particular situations, where the system is forced to reorganize or adapt to new situations. This is where the self-* mechanisms come into play and can be tested separately.

Keywords: Self-Organizing Systems, Adaptive Systems, Software Engineering, Software Testing, Multi-Agent Systems, Quality Assurance

1 Introduction

For many years, nature has been a source of inspiration for the design of information systems, e.g., in the research areas of artificial intelligence, multi-agent systems, and self-organizing systems. The core idea is to develop systems by using the so called self-* properties (self-healing, self-configuring, and self-optimizing), which correspond to biological mechanisms. The foundation of many of the self-* properties is self-organization (SO). SO enables a restructuring of the system and its components in order to conform to its environment to fulfill its goals. This new flexibility leads to challenges in engineering these systems. For example, the behavior can not be specified fully at design-time and adaptation decisions are moved to run-time. This makes it hard to assure that those systems fulfill their requirements, but even more necessary to take appropriate measures. In this connection testing plays a decisive role. The pivotal question is how to handle the state space of a self-organizing, adaptive systems (SOAS) which is developing in an evolutionary fashion during the execution of the system.

Our approach for testing is based on the *Corridor Enforcing Infrastructure (CEI)*, an architectural pattern for SOAS. The CEI uses the concepts of feedback-loops to continuously observe and control the system if necessary. On that account monitoring is used to achieve situational awareness which is prerequisite to organize the system in a way to fulfill its requirements. The bulk

of concepts and techniques used in the CEI for controlling and organizing the system is triggered by the violation of constraints. An error-free CEI will consequently guarantee a system that fulfills its requirements at every time. On this account, we claim that testing SOAS can be achieved by testing the CEI. Here we benefit from specific characteristics: we are able to reduce the relevant state space of the system to be tested tremendously and to gain a clear distinction between correct and incorrect behavior out of the concepts of the CEI—essential for evaluating the tests. SOAS are highly distributed systems—making them hard to test—and are interacting locally to solve problems. This locality is exploited in our approach, because many test cases focus on single agents or small agent groups of which the system is composed; this focus make it easier to execute and evaluate them.

In the following sections we will discuss related work and describe our approach for testing SOAS, which is based on concepts of the CEI that are introduced in Sec. 3 and concepts to reveal failures out of the CEI in Sec. 4.

2 Related Work

The necessity of testing adaptive systems has been recognized in the testing community [8, 9, 12] as well as in the community of adaptive systems [5, 11]. Both run-time and design-time approaches have considered indeterminism and the emergent behavior as main challenges for testing adaptive systems.

Run-time approaches for testing take up the paradigm of run-time verification [3, 7, 4]. They shift testing into run-time to be able to observe and test, i.a., the adaption to new situations. Camara et al. [1] are using these concepts to consider fully integrated systems. The authors simulate the environment of the system to investigate its adaptive capabilities, and to collect data about how the system reacts. The sampled data is then combined with a probabilistic model of the system and used as input for a model checker in order to evaluate given system properties. Similar approaches are taken by Ramirez et al. [10]. They use the sampled data from a simulation to calculate a distance to expected values derived from the goal specification of the system.

Overall the run-time approaches are limited to test the fully integrated system and are only sampling, but not evaluating, data at run-time. We extend this simulation testing of a fully integrated system by a bottom-up testing strategy, starting from single agents, in order to cover testing in the whole software development life cycle. Furthermore, we are able to evaluate the runs online without complex model checking on the system level with the usage of the CEI.

Design-time approaches for testing adaptive systems [8, 9, 12] are focusing on specific subsets, e.g., the implementation of single agents. Zhang et al. [12] are focusing on testing the execution of plans of single agents and groups of agents. This focal point does not allow to evaluate adaptive or even self-organizing characteristics of the system, contrary to our approach.

In conclusion, testing adaptive systems is in focus of some research groups. However, none of them is extending the techniques to self-organizing systems

which are able to form emergent behavior. We aim to deal with SO in all its facets and therefore combine run- and design-time testing techniques.

3 The Corridor Enforcing Infrastructure: Enabler for Systematic, Automatic Testing of SOAS

The CEI is an architectural pattern for SOAS, based on decentralized feedback-loops used to monitor and control single agents or small groups of agents in order to enforce that all system requirements are fulfilled at all times. The decentralization follows from a modular system structure where the system requirements are addressed to single or small groups of autonomous agents. An error-free CEI makes it unnecessary to test concrete system states, because the CEI guarantees that all states fulfill the system requirements. Consequently, only the CEI and its mechanisms must be tested instead of the whole SOAS en bloc, this significantly reduces the test effort.

The CEI is based on the concept of the corridor of correct behavior (CCB) which is formally introduced by Güdemann et al. [6]. The corridor is formed by all requirements, realized as constraints, of the considered system. An exemplary corridor is shown in Fig. 1. If all constraints are fulfilled, the system is inside the corridor. Otherwise, the system leaves the corridor, indicated by a flash in Fig. 1, this demands that the system is transitioned into a safe-mode. Furthermore, the system has to be reorganized to return to a state within the corridor; this is shown through the check mark in Fig. 1. An error would occur if a transition outside of the corridor is taken, indicated by a cross in Fig. 1. The CEI implementation

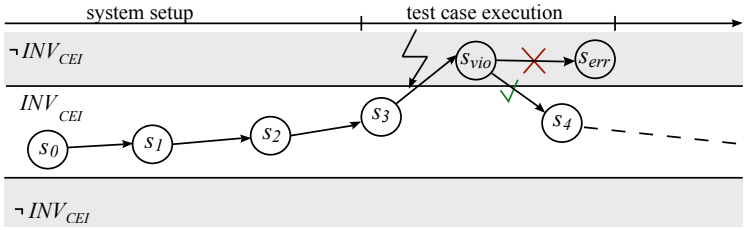

Fig. 1. Schematical state-based view of the corridor of correct behavior and the different phases of testing SOAS. INV_{CEI} is the conjunction of all constraints of the system controlled by the CEI.

of the CCB is based on a monitoring and controlling infrastructure. The monitoring infrastructure observes the system and detects a violation of the corridor; in some cases it can be (semi-)automatically generated from the requirement documents of the system, as shown by Eberhardinger et al. [2]. The controlling infrastructure ensures that the system is transitioned into a safe mode when the constraints are violated and reorganizes the system that all requirements are fulfilled again.

The CEI and its mechanisms will be tested in order to validate that the SOAS fulfills all requirements at all times. Several failures can occur independently due to the SO mechanisms used in the controlling infrastructure, e.g., genetic algorithms or constraint solving. Our SOAS testing approach examines special situations where the system is confronted with environmental situations in which SO or adaptation is required and the CEI stressed. In the next section we will outline how to cope with this in our testing concepts in more detail.

4 Environmental Variation Scenarios: Forcing the CEI to Reveal Failures

To force the CEI to reveal failures we use test scenarios which stress this infrastructure. We apply a compositional test strategy which is geared to the classic testing strategies unit testing, integration testing, and system testing. Thus, we are able to form a test process for the whole development cycle.

The nature of the mechanisms within the CEI are on adapting and reorganizing the system to react to its changing environment. Thus, we aim to automatically generate so called environmental variation scenarios (EVS) for testing the mechanisms of the CEI. For a *single agent*, this means in fact we want to validate that changes in the environment are detected via sensors used by the monitoring infrastructure and the agent adapts accordingly to fulfill its goals. The input for the agent can be simulated by mocking in the test system without setting up complex initial states. We automatically extract the EVS from the specification of the agents and use them for testing. For this purpose we have to take a closer look at the behavioral model (e.g., state machines) of the agents to derive a model of their interactions with the environment. To build the EVS from these models we use model checking techniques that find sequences in which the interaction between the agent and its environment leads to a violation of the CCB, illustrated by a flash in Fig. 1. Such a violation has to be detected by checking the system state according to the CCB through the monitoring-infrastructure. It must be pointed out that such violations are no failures of the CEI, but the CEI has to detect them and take the adequate actions.

The EVS are, as introduced, based on single agent situations. This means a fine-grained level of decomposition which also reflects the way the constraints of the CCB are defined and consequently the concepts of the CEI. To be able to test scenarios where several agents are interacting, we need to extend the EVS. Interactions are essential in testing, especially to cover SO mechanisms, since the results of these mechanisms mainly depend on interactions of the participating agents, which lead to restructuring and adaptation of these participating agents. This characteristic has to be reflected in the test scenarios for groups of agents on the *agent interaction level*. Interactions between agents in SOAS use different mechanisms, e.g., stigmergy or simple message passing. These interdependencies have to be taken into account to derive a scenario for a group of agents as well as for a fully integrated system under test. The interactions can be culled from the communication protocols of the system. The scenarios for single agents can

now be connected via the possible interactions between agents. The result will be an interaction model for selected scenarios. Because of evolutionarily changing system structures, it is hard to predict which communication scenarios will be executed. But by focusing on the EVS for derivation of test cases, the resulting interaction scenarios are just small subsets of the overall system communication and therefore much more manageable in testing. In contrast to the tests for single and separated agents, the execution of EVS for groups of interacting agents can, in general, not be executed without a setup phase, because the mocked up environment is increasingly replaced by real, interacting agents. The needed structures or parameter settings of the mechanisms have to be initialized in a setup phase. This reflects the fact that the SO mechanisms are sometimes based on evolutionary algorithms, e.g., genetic algorithms can be used for calculating new interaction structures in an agent group and therefore the needed edge weights have to be trained initially before the execution of a test case.

Subsequent to single agent and agent interaction the next step is—analogously to classical system testing—to test the *fully integrated system*. All mocked-up components will be replaced by real agents. To minimize the overhead for setup we combine test cases so that they can be executed in sequence. In order to connect the scenarios it is necessary that the system state of a completed test case matches with the needed system state a next test case. We use search-based techniques to retrieve the matching scenarios. Furthermore, it is of interest to test interleaving scenarios in the fully integrated system to evaluate interdependencies of the self-* mechanisms within the CEI. To form these scenarios we combine agents with the following specifics: agents which know each other, agents which are able to communicate with each other, and agents which cooperate.

Besides developing test strategies we aim at establishing a set of coverage metrics which are focused on the coverage of the CEI. These metrics are reflecting our approach on testing SOAS by using the CEI. Standard coverage metrics make statements on code fragments, logical elements in the code, or on control/program flow coverage. For our testing approach the significance of classical coverage metrics is no longer sufficient. Based on the concepts of input space partitioning we describe the expressiveness characteristics for the input domain. The metrics based on partitioning consequently give evidence on the coverage of each region separated by the defined characteristics. As already stated above the most important characteristics for the mechanisms in the CEI are based on changing environmental situations and interactions between agents. Consequently, we will use these features to partition the input space. Based on this partitioning different kinds of combinatorial metrics can be retrieved and used for SOAS. The resulting metrics are able to cope with indeterministic, evolutionary structures of SOAS by benefiting from the CEI concepts.

5 Conclusion & Outlook

We outlined an approach to cope with the complexity which arises from the flexibility of SOAS. For this purpose, we briefly introduced the architectural

pattern called CEI that enables systematic, (semi-)automatic testing of SOAS. The concept of the CCB is used to form EVS. We generate these scenarios to accelerate the system to adapt and reorganize to new situations in order to investigate the CEI. To make a statement about the adequacy of the performed tests and to select concrete test cases, we introduced metrics in the context of the CEI. Overall, this constitutes a research agenda towards testing self-organizing, adaptive systems.

References

1. Camara, J., Lemos, R.D.: Evaluation of resilience in self-adaptive systems using probabilistic model-checking. In: Proc. 2012 ICSE Wsh. Software Engineering for Adaptive and Self-Managing Systems (SEAMS'12). pp. 53–62 (2012)
2. Eberhardinger, B., Steghöfer, J.P., Nafz, F., Reif, W.: Model-driven synthesis of monitoring infrastructure for reliable adaptive multi-agent systems. In: Proc. 24$^{\text{th}}$ IEEE Int. Symp. Software Reliability Engineering (ISSRE'13). pp. 21–30. IEEE (2013)
3. Falcone, Y., Jaber, M., Nguyen, T.H., Bozga, M., Bensalem, S.: Runtime verification of component-based systems. In: Barthe, G., et al. (eds.) Proc. 9$^{\text{th}}$ Int. Conf. Software Engineering and Formal Methods (SEFM'11). pp. 204–220. No. 7041 in Lect. Notes Comp. Sci., Springer (2011)
4. Filieri, A., Ghezzi, C., Tamburrelli, G.: A formal approach to adaptive software: Continuous assurance of non-functional requirements. Formal Asp. Comp. 24(2), 163–186 (2012)
5. Fredericks, E.M., Ramirez, A.J., Cheng, B.H.C.: Towards run-time testing of dynamic adaptive systems. In: Proc. 8$^{\text{th}}$ Int. Symp. Software Engineering for Adaptive and Self-Managing Systems (SEAMS'13). pp. 169–174. IEEE (2013)
6. Güdemann, M., Nafz, F., Ortmeier, F., Seebach, H., Reif, W.: A specification and construction paradigm for organic computing systems. In: Brueckner, S.A., Robertson, P., Bellur, U. (eds.) Proc. 2$^{\text{nd}}$ IEEE Int. Conf. Self-Adaptive and Self-Organizing Systems. pp. 233–242. IEEE Computer Society (2008)
7. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebr. Program. 78(5), 293–303 (2009)
8. Nguyen, C.D.: Testing Techniques for Software Agents. Ph.D. thesis, Università di Trento (2009)
9. Padgham, L., Thangarajah, J., Zhang, Z., Miller, T.: Model-based test oracle generation for automated unit testing of agent systems. IEEE Trans. Softw. Eng. 39(9), 1230–1244 (2013)
10. Ramirez, A.J., Jensen, A.C., Cheng, B.H.C., Knoester, D.B.: Automatically exploring how uncertainty impacts behavior of dynamically adaptive systems. In: Alexander, P., et al. (eds.) Proc. 26$^{\text{th}}$ IEEE/ACM Int. Conf. Automated Software Engineering (ASE'11). pp. 568–571. IEEE (2011)
11. Wotawa, F.: Adaptive autonomous systems — from the system's architecture to testing. In: Hähnle, R., et al. (eds.) Leveraging Applications of Formal Methods, Verification, and Validation, pp. 76–90. Comm. Comp. Inf. Sci., Springer (2012)
12. Zhang, Z., Thangarajah, J., Padgham, L.: Model based testing for agent systems. In: Decker, et al. (eds.) Proc. 8$^{\text{th}}$ Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009). pp. 1333–1334. IFAAMAS (2009)