Model-based, Adaptive Testing of Organic Computing Systems

Benedikt Eberhardinger
Institute for Software & Systems Engineering, University of Augsburg, Germany
benedikt.eberhardinger@informatik.uni-augsburg.de

***Abstract*—Testing is one of the critical points in the software engineering process, especially in highly complex, autonomic systems. Yet there is no clear concept how to test an Organic Computing System in an appropriate way. The main problem in this field is to handle the self-organizing and adaptive behaviour of those systems. I propose a model-based and adaptive approach to test Organic Computing Systems.**

***Index Terms*—Model-based Testing; Adaptive Testing; Testing; Organic Computing; Multi-agent System**

I. Introduction

Organic Computing (OC) Systems are large, distributed, heterogeneous systems (mostly multi-agent systems), which are aware of their environment and themselves in order to autonomously organize and adapt to achieve certain goals [1]. For that reason, the agents are communicating with other agents to cooperate and interact. This leads to an evolutionary changing system. Due to the fact that the environment and the behaviour of other agents are hard to predict, unforeseeable system states can occur. This fact makes it difficult, but even more necessary, to design convincing tests to detect unintended behaviour of the OC System.

The complexity of testing self-organizing, adaptive systems could be spilt into two major facts: On the one hand, the self-organizing and adaptive characteristics make it tough to design and perform suitable tests. On the other hand, testing distributed, concurrent software is still an awkward challenge [2]. While the issues of testing distributed, concurrent software have been a problem in research for a long time [3], the difficulties of testing OC Systems add a new view. An example is the task to test the aspect of reorganization. Therefore, the test has to deal with the interactions within the OC Systems, because a task could be achieved in many different ways that could radically change by effects of other agents and the environment. For example, it is pretty straight forward to test whether or not a single agent achieves a desired output under a given circumstance. But if the task is allocated to a system of agents which, e.g., could form coalitions for achieving this goal it is rather difficult to say which agent performs correctly or not, which in this example requires to know the responsibilities within the coalitions.

To cope with this complexity, a structured process is needed. The aim is to design a model-based, adaptive approach designed for OC Systems. For that reason, existing techniques for distributed, concurrent software systems are extended for OC Systems. The here presented work of my doctoral dissertation is in an early stage, thus, this paper presents only a crude outline on this topic: First, related work from the field of model-based testing distributed, concurrent systems as well as dynamic symbolic execution for testing is presented in Sect. II. Afterwards the challenges in testing OC Systems are presented in Sect. III. To cope with these challenges my approach is outlined in Sect. IV.

For the further explanations a short, simplified case study of a distributed power management system will be used [4]. The power plants in this system are partitioned into Autonomous Virtual Power Plants (AVPPs). These AVPPs coordinate power plants in order to meet power demands. Each AVPP is represented by an agent and has to fulfil a specific power output to accomplish the global goals of the system. Furthermore an AVPP can recognize a deviation of the global output and react to this in cooperation with the other AVPPs. The demand for each agent is allocated autonomously among the AVPPs. To simplify matters, all AVPPs have the same capabilities, e.g., they all have the same maximum power output in production, same reaction times and same costs.

II. Related Work

In the area of model-based and automatic testing of concurrent, distributed systems different approaches have been introduced and applied. These concepts are partly related to the approach of model-based, adaptive testing of OC Systems. The most important of them will be introduced below.

There has been significant research in the area of model-based testing. Zander et al. [5], Broy et al. [6] and Utting et al. [7] provide an overview of the research in this field. The basic idea of model-based testing is to develop a formal model of the system under test (SUT) and to use this model to generate tests for the system automatically. To obtain this model the requirements and functional models of the SUT are combined to encode the intended behaviour of the system. Therefore, different kinds of models are applied, which describe the tests as well as the intended behaviour [5], [6]. These models have to describe all possible different system transactions. In OC systems it is inpossible to define all possible system transactions. Thus, a model has to be able to be more flexible to support testing of OC Systems. By defining the model in terms of a corridor of correct respectively intended behaviour this flexibility could be gained. My approach is to introduce a goal-oriented behaviour description in the system requirements.

This enables the static model to be more flexible according to the SUT.

Another problem, which has gained special importance in the case of goal-oriented behaviour descriptions, is to generate automatically good test case and the corresponding test oracles. Godefroid et al. [8] addressed this issue with the tool "Directed Automated Random Testing" (DART). The authors combine three main techniques: First the interfaces of a program to its environment are extracted automatically. Afterwards a test driver is generated for all interfaces to perform random tests in order to simulate the environment. Last the output of the program is dynamically analysed how it behaves under random conditions. Based on this results new test paths are generated. By using these techniques, DART is able to test a program that compiles – as a black-box – without writing any test driver. In other words, DART constructs well-formed random inputs for the system, which are used to stress-test the system. On the basis of DART Godefroid et al. [9] developed the SAGE system, which allows white-box fuzz tests. The program is executed, beginning from a defined point in the program code, with a fixed initial input. Afterwards the algorithm of SAGE is gathering input constraints from conditional statements. These collected constraints are used to generate test inputs.

In the field of symbolic execution tests other efficient approaches are made by Rungta et al. [10], Davies et al. [11] as well as Griesmayer et al. [12]. Griesmayer et al. [12] especially take the problem of testing distributed objects into account.

In theory systematic dynamic test generation can lead to full program path coverage. Furthermore it allows the test suite to adapt to the program. But, the problem of these approaches is the lack in scalability, because the techniques could lead to a path explosion (cf. [13]). My presented approach tries to cope with this problem by decomposition.

An alternative way to find the input for the test cases is to use a model checker that generates possible inputs out of a given model. Gargantini et al. [14] showed an approach that is generating test sequences with a model checker. My approach is trying to combine these techniques with the dynamic symbolic execution of tests.

To apply these related work into the approach of testing OC Systems it is necessary to take the fact of distribution and concurrency into account. In the area of testing distributed, concurrent systems a lot of research has been done. Souza et al. [2] gives an overview of the most resent research in this area. These techniques have to be taken into account to be able to test OC Systems adequately, as mentioned in a later section of this paper.

Concluding, in different related areas of research already techniques are developed which could be partly integrated in this approach. Indeed, there is no related work in the area of testing OC Systems. Testing adaptation, self-organization, etc. is an unexplored area, where this work tries to give its contribution.

III. Challenge

In general, testing is "*[the] process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component*" [15]. The intention of the evaluation is to identify a failure, which is an event that leads to a state of the observed system which does not perform a required function within specified limits [16]. To detect this failure of the observed system, it is necessary to be able to state whether or not the system is in a correct state.

In OC Systems, this is not always straightforward to answer. One approach, made by Schmeck et al. [17] is to separate the global state space of an OC System into a *target space*, an *acceptance space*, a *survival space* and the so called *dead zone*. In this context a failure will occur if the system state is in the *dead zone* and so is "unrepairable" by itself through reconfiguration. Therefore, tests have to check in which space the system is. In this context the following challenges for testing OC Systems could be identified:

- Deal with reconfiguration and adaptivity
- Handle interleaved feedback-loops
- Cope with distribution and concurrency
- Find a clear classification of failures for OC Systems
- Create an appropriate environment for testing
- Define suitable levels of testing OC Systems

A. Reconfiguration and Adaptivity

Reconfiguration and adaptivity allows OC Systems to recover from system states which are not fulfilling the requirements. This situation therefore does not have to be a failure, but it is important that a system which is in a reconfiguration phase still behaves in an acceptable and safe manner. In the example of the AVPPs this case appears if one agent fails to gain the demanded output. But all other agents recognize this deviation and therefore produce together more output as set to each of them. This leads to an achievement of the global goal and therefore no failure despite of a local deviation. So a local deviation (which is not part of the goal) does not lead to a failure, because the system reconfigures and reacts adaptively on this situation.

B. Interleaved Feedback-Loops

The behaviour of the self-organizing and adaptive agents of an OC System could be described by a number of interleaved feedback-loops [17]. As a consequence, it is possible that all agents of the system perform as required, but the whole system fails, e.g., all AVPPs recognize a deviation of the global goal and therefore adapt their own output, because every agent is trying to fix the deviation on its own, the global system overreacts. Consequently every agent performs locally correct by reacting to the new situation, but the system fails with respect to its goal. Thus, one important challenge is to cope with these interdependencies in OC Systems.

C. Distribution and Concurrency

In addition, there are the challenges given by the distribution and concurrency of the systems. Namely deadlocks, livelocks, and data races have to be identified. These are mainly situations where a concurrent usage of a resource leads to a failure.

D. Classification of Failures

Despite the known aspects about possible failures in distributed, concurrent systems and the outlined failures of OC Systems, there is no clear classification of failures for OC Systems. So an important step towards testing OC Systems must be to take this problem into account.

E. Appropriate Environment for Testing

Building up on a classification of failures for OC Systems, it is necessary to consider how to find these possible failures in a concrete system. As a result of the self-organization and the adaptation of an OC System, because of other possibly not controllable agents and the environment of the system, it is difficult to force the system to perform in a way that allows a concrete evaluation. It is indispensable to create an environment for the tested part of the system to generate reproducible results of the evaluation for a concrete requirement. This includes a complete initial configuration with possibly required mock-ups or stubs. For testing the system of AVPPs it is obvious that such a test-suite is inevitable. Mock-ups have to simulate the energy grid, the consumers and other participants to perform suitable evaluations. Furthermore, the adaptivity of a single AVPP should be tested separatly from the other AVPPs to perform reproducible, independent results. For example to test the reaction of an AVPP to a deviation of the global demand and the global production other agents could influence the results and so it is difficult to evaluate this. To perform tests in an appropriate way, a generic approach has to be found for setting up tests in the described way. As a part of this generic approach the differentiation between testing and simulating a system has to be done in a more clear way.

F. Suitable Levels of Testing

Depending on the requirements to be evaluated, tests have to be performed on an appropriate level. For example, tests must be performed for single agents or for groups of agents to observe interdependencies. In the example of the AVPPs there could be one level to test a single AVPP separately, one other to test all controllable AVPPs together and on another level with non-controllable agents. Furthermore there could be several levels of the environment, e.g., the Bavarian energy market, the German energy market, the European energy market and so on. Different levels could lead to different interdependencies and so exhibit different kinds of failure. Common levels of testing are unit tests, integration tests, system tests, and acceptance tests. But OC Systems are, as already shown, different from classical software systems. So the levels have to be redefined and possibly extended according to the dedicated needs.

IV. Approach

To cope with the challenge to test OC Systems adequately, the intention is to develop a model-based, adaptive approach with the following main concepts:

- Decomposition
- Discover sequences of violation
- Model-based design and execution of tests

A. Decomposition

To achieve an adaptive test, techniques of dynamically creating and executing symbolical tests, are used to build up on ([13], [9]). These techniques lead to a high coverage by trying to explore every possible path for a symbolic input automatically. One problem to be solved here is the scalability of this approach, because the number of possible execution paths is exponential in the size of the input [13]. Especially in an OC System this could get problematic, because of its complexity and size. The OC System of the AVPPs, e.g., could represent the whole energy market of Europe, which means an enormous amount of participating agents. For this reason, decomposition plays a significant role. The approach by Steghöfer et al. [18] shows how to use and decompose goal-oriented requirements for monitoring OC Systems. For this purpose a set of constraints for specific agents is defined, which describes the global invariant of the system. This decomposition could be reused in the tests.

Built up on decomposed test cases the scenario has to be extended ”bottom-up“ to even get the possibility to test self-organization, adaptivity and other features of distributed, concurrent systems.

B. Sequences of Violation

Furthermore, it is important to discover the sequences for a possible violation and thus to reduce the size of the paths to be tested. A possibility is to use techniques of model checking to generate sufficient test cases. To enable these process I propose to use techniques build up on model-based testing.

Despite the problem of creating the appropriate test cases, it is important to get a good test oracle for the evaluation. The Restore Invariant Approach (RIA) defines the correct behaviour of an OC System by a corridor of correct behaviour [19]. The corridor of the RIA is based on a global system invariant, which is a conjunction of all constraints of a system. Based on a system trace it is therefore possible to decide whether or not the system is performing correct or has to be reconfigured. This view on correct and incorrect behaviour of an OC System could be adapted as a test oracle for the evaluation. For the example of the AVPPs the system invariant only consist of one constraint, which is namely that the demand equals to the consumption, so every evaluation has to check this characteristic of the system.

C. Model-based Design and Execution of Tests

In this approach models are used to design and execute test cases. For that reason models represent the required behaviour

of the system and its environment. Constraints designed for an OC System could be, as already shown, used to model the required behaviour of the system. Also there is a need of some relies according to the environment to specify the test. A related approach to this modelling is made in the concept of Rely/Guarantee (R/G), which is used by Nafz et al. [20] to formally model and verify OC System. In the case of the AVPP case study the model for the test would be the constraint of balanced production and demand. The rely according to the environment is that the demand would not be bigger than the maximum productivity of the AVPPs. Beside the required behaviour and the relies there is a necessity for structural and behaviour models of the tested components. Therefore, models from the *UML* standard, like activity and class diagrams are obvious to use. These models now could be used to build concrete test cases and a test environment.

D. AVPP Case Study

The global invariant of the AVPP System is that demand is equal to the production of the system, a pretty easy way to break this down to constraints for single agents would be to divide the global demand by the number of AVPPs. Thus on the lowest level each single AVPP could be tested separately from the system. For that reason the test-environment has to mock-up the other agents, which perform for one test the correct and for another the incorrect output. Furthermore there has to be a mock-up, which simulates the consumer. After that the evaluation just has to check whether or not the output of the agent is as high as expected. The expectation is that the agent produces as much output as needed so that the sum of simulated production and its own production equals to the simulated demand. Of course the agent is limited to its maximum productivity, this has to be taken into account for the test cases in form of a rely. In the next case two Agents of the same kind could be tested together in a pretty similar environment to evaluate the interaction, e.g., the reaction on a deviation.

V. Conclusion

Self-organizing and adaptive properties of Organic Computing (OC) systems are challenging research topics with a lot of open issues to be solved (cf. [21]). This paper points out the major challenges in testing OC Systems. My approach is to cope with these problems by using a model-based and adaptive concept. To apply this, useful techniques and processes from the field of testing distributed and concurrent systems [2] will be adapted. This could be combined with work in the field of dynamic creating and executing symbolical tests [13], to achieve a certain kind of adaptivity. The basis for testing is a model, which can be processed. The related work in model-based testing (especially for reactive systems [6]) could give a good initial point.

The aim of the future work is to provide a test suite with an according process to test OC systems.

References

[1] C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds., *Organic Computing - A Paradigm Shift for Complex Systems*. Springer, 2011.

[2] S. R. S. Souza, M. A. S. Brito, R. A. Silva, P. S. L. Souza, and E. Zaluska, "Research in concurrent software testing: a systematic review," in *Proc. Wsh. Parallel and Distributed Systems: Testing, Analysis, and Debugging*, ser. PADTAD '11. ACM, 2011, pp. 1–5.

[3] C.-S. D. Yang, "Program-based, structural testing of shared memory parallel programs," Ph.D. dissertation, University of Delaware, 1999.

[4] G. Anders, F. Siefert, J.-P. Steghfer, H. Seebach, F. Nafz, and W. Reif, "Structuring and controlling distributed power sources by autonomous virtual power plants," in *Proc. of the IEEE Power and Energy Student Summit*, ser. PESS '10. IEEE, 2010.

[5] J. Zander, I. Schieferdecker, and P. J. Mosterman, Eds., *Model-Based Testing for Embedded Systems*. CRC Press, 2012.

[6] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, *Model-Based Testing of Reactive Systems*, ser. LNCS. Springer, 2005.

[7] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Softw. Test. Verif. Reliab.*, vol. 22, no. 5, pp. 297–312, Aug. 2012.

[8] P. Godefroid, N. Klarlund, and K. Sen, "Dart: directed automated random testing," in *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, ser. PLDI '05. New York, NY, USA: ACM, 2005, pp. 213–223.

[9] P. Godefroid, M. Y. Levin, and D. A. Molnar, "Automated whitebox fuzz testing," in *Proc. 16th Annual Network and Distributed System Security Symposium*, ser. NDSS '08. Internet Society, 2008.

[10] N. Rungta, E. G. Mercer, and W. Visser, "Efficient testing of concurrent programs with abstraction-guided symbolic execution," in *Proc. 16th International SPIN Workshop on Model Checking Software*. Springer-Verlag, 2009, pp. 174–191.

[11] M. Davies, C. Psreanu, and V. Raman, "Symbolic execution enhanced system testing," in *Verified Software: Theories, Tools, Experiments*, ser. LNCS, R. Joshi, P. Mller, and A. Podelski, Eds. Springer, 2012, vol. 7152, pp. 294–309.

[12] A. Griesmayer, B. Aichernig, E. B. Johnsen, and R. Schlatte, "Dynamic symbolic execution for testing distributed objects," in *Proc. 3rd International Conference on Tests and Proofs*, ser. TAP '09. Springer-Verlag, 2009, pp. 105–120.

[13] C. Cadar, D. Dunbar, and D. Engler, "Klee: unassisted and automatic generation of high-coverage tests for complex systems programs," in *Pro. 8th USENIX Conf. Operating systems design and implementation*, ser. OSDI'08. USENIX Association, 2008, pp. 209–224.

[14] A. Gargantini and C. Heitmeyer, "Using model checking to generate tests from requirements specifications," in *Proc. 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering*, ser. ESEC/FSE-7. Springer-Verlag, 1999, pp. 146–162.

[15] IEEE, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std. 610.12-1990.

[16] ——, *IEEE Standard Classification for Software Anomalies*, IEEE Std. 1044 - 2009.

[17] H. Schmeck, C. Müller-Schloer, E. Çakar, M. Mnif, and U. Richter, "Adaptivity and self-organization in organic computing systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 5, no. 3, pp. 10:1–10:32, Sep. 2010.

[18] J.-P. Steghöfer, B. Eberhardinger, F. Nafz, and W. Reif, "Synthesis of observers for autonomic evolutionary systems from requirements models," in *Proc. 13th IFIP/IEEE Symposium on Integrated Network and Service Management*, ser. IM '13. IEEE Computer Society, 2013.

[19] F. Nafz, H. Seebach, J.-P. Steghöfer, G. Anders, and W. Reif, "Constraining self-organisation through corridors of correct behaviour: The restore invariant approach," in *Organic Computing A Paradigm Shift for Complex Systems*, C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds. Springer, 2011, pp. 79–93.

[20] F. Nafz, J.-P. Steghfer, H. Seebach, and W. Reif, "Formal modeling and verification of self-* systems based on observer/controller-architectures," in *Assurances for Self-Adaptive Systems*, ser. LNCS, J. Cmara, R. Lemos, C. Ghezzi, and A. Lopes, Eds. Springer, 2013, vol. 7740, pp. 80–111.

[21] C. Nguyen, A. Perini, C. Bernon, J. Pavn, and J. Thangarajah, "Testing in multi-agent systems," in *Agent-Oriented Software Engineering X*, ser. LNCS, M.-P. Gleizes and J. Gomez-Sanz, Eds. Springer, 2011, vol. 6038, pp. 180–190.