Approaching Performance Testing From a Model-Based Testing Perspective

Fredrik Abbors, Dragoş Truşcan
Department of Information Technologies, Åbo Akademi University
Joukahaisenkatu 3-5 A, 20520, Turku, Finland
Email: {Fredrik.Abbors, Dragos.Truscan}@abo.fi

***Abstract*—The paper introduces the concept of *model-based performance testing*, which we plan to pursue in our research. The underlying idea is to describe various performance aspects as well as functional aspects of a software system using modeling languages like UML, and from the resulting models to automatically design tests that can be used for performance testing. In our research, we also plan to focus on how the modeling and traceability of performance requirements can be achieved across the testing process.**

***Keywords*-Model-Based Testing, Performance Testing, Requirements Traceability**

I. Introduction

The complexity of software systems and applications are increasing. Additionally, software systems are usually deployed within different types of distributed environments. Performance characteristics such as throughput, response time, and scalability are becoming increasingly more important for such applications and systems. For this reason, it is critical to verify that the system satisfies its performance requirements. Studies [1] show that a significant percentage of the deployed applications have in practice performance issues. These constitute one of the major fault categories in the telecommunications domain [2].

Performance testing is one of the activities of the Software Performance Engineering process as defined by Smith and Williams [3]. Performance testing is usually carried out manually in the later stages of development and constitutes a tedious and time consuming task. Having a structured and automated approach where performance test are derived from abstract model representations of the system would allow to leverage the effort of manually designing performance tests. In order to have a successful automated performance testing process, tests need not only to be generated automatically, but also executed and the results of the test execution analyzed and interpreted. In our opinion, the approach would benefit even more if standard modeling notations like the Unified Modeling Language (UML) [4] are used.

The paper will proceed with a short introduction to model-based testing in Section II, followed, in Section III, by a overview of related work in the field. Then, Section IV will discuss performance testing and its challenges, whereas in Section V we describe our approach in combining the model-based testing and performance testing.

II. Model-based Testing

Model-Based Testing (MBT) [5] is a testing paradigm, in which test cases are derived from an abstract model that describes the functional behavior of the System Under Test (SUT). In many occasions, the behavioral model is complemented with other specifications like data, architecture. etc. The reason for having an abstract model is that one usually does not want to test the whole system, but rather focus on a particular part, leaving out the information that is not relevant at a given abstraction level. Thus, the way MBT uses abstraction makes it suitable for different phases of the testing process starting from unit testing to system testing.

In traditional testing, test cases are written manually by analyzing the requirements. In MBT, a *test model* describing the expected behavior of the SUT is initially developed. Once the model is complete, test cases are automatically designed by traversing the model based on selected *coverage criteria*. The resulting test cases are abstract and have to be brought to the level of abstraction understood by the SUT.

The time to develop a test model should be shorter than developing all the test cases by hand. In practice, the time to develop a test model depends of course on the skills of the tester and the complexity of the system. An inexperienced tester may develop the test model in the same time as it would take for an experienced tester to write all the test cases manually. MBT is not only an activity for automatic test generation; it also includes automatic execution and analysis of the tests. These tasks make MBT even more challenging than just automatic test generation. Research in the area of MBT has been carried out and is still going on, investigating how to apply techniques and tools to automate the testing process.

There are many advantages using MBT. One of the strongest advantages is that large numbers of test cases can be generated in a relatively short amount of time. Another advantage of MBT is the maintenance of the test cases. As the project evolves the requirements may change, the test cases also need to change. If one uses manual testing methods, then all the created test cases must be examined to verify that they are still valid i.e., they are not affected by the changed requirements. In MBT one could simply apply the changes in the system model and regenerate all the test cases.

III. Related Work

Most of the performance modeling techniques, that we have reviewed, are used for development, simulation, maintaining, and analysis of performance characteristics in software systems, but only a handful are actually used for generation of performance tests. There are existing performance testing techniques [6][7][8] that are actually used to derive performance tests from models, but very few use UML as domain language.

Most of the techniques use a domain specific language to capture the performance characteristics of the system under test (SUT) from where they automatically or manually derive performance tests. We plan to elaborate on the existing techniques and apply them to our approach.

One of the advantages of using UML as modeling language for performance test generation is the graphical expressiveness of the language. This, together with abstraction, implies that the resulting models are easy to maintain and understand. Another advantage of using UML for performance testing is that UML is becoming a standard notation for software and hardware systems modeling in many industries [9][10][11]. Besides, UML allows testers to embed both static and dynamic aspects of the system into the models, using different diagram types. These characteristics make UML suitable for systematic performance evaluation as well. Research has been done on creating and maintaining UML performance models for various distributed systems [12][13][14][15][16]. In addition, UML can be customized to serve specific purposes. Standardized UML profiles i.e., UML Profile for Schedulability, Performance, and Time (SPT) [17], UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [18], or UML Testing Profile (UTP) [19] have already been developed and they allow testers to model performance related characteristics of the system.

IV. Performance testing and challenges

Performance testing is a way to evaluate the design of the system with respect to performance requirements. Performance testing is also the means to determine how fast some critical aspects of a system perform under a particular load. With a sound and stable performance testing plan one can measure different characteristics of a system, for instance, the response times or detect possible workload thresholds for the system. Performance testing can also demonstrate the reliability and scalability of the system, and even measure the systems resource usage. All these factors play an important role in the today's software systems.

Performance testing is usually performed right before the actual users start using the system. One of the biggest challenges with performance testing today is *replicating a similar environment* (for testing purposes) as the environment, in which the system is going to be deployed. Maintaining such an environment may in some cases be impractical, too expensive, or not even possible. The solution to this is to scale down the testing objectives based on the capacity of the available testing environment.

Other challenges that performance testers are facing is the generation of appropriate test data, the identification and specification of *proper performance requirements*, and the *establishment of performance testing goals*. Generating meaningful and large quantities of test data for performance testing is not a trivial task, for example, generating test data from a system with billions of users with multiple user profiles, user names, passwords, etc. Before starting performance testing, one needs to have a stable enough software to test so that the results are conclusive and not influenced by, for example, a software crash.

Work conducted in [20][21] listed several challenges for performing performance testing:

- Capturing performance requirements at the beginning of the development process;
- Development of coverage measures for performance tests;
- Model-assisted test design and evaluation.

Performance testing is usually *carried out manually* by identifying appropriate functional tests that are executed against the system in various quantities in order to determine the responsiveness of the system. Identification of test results and the detection of possible bottlenecks in the system is also in many cases preformed manually and is therefore considered a tedious task. Hence, there is clearly a need of having automated ways to generate, execute, and validate performance tests for various distributed system. Model-based testing could be a way to address these problems.

V. Model-Based Performance Testing

As performance testing is often carried out during the later stages of development, we will try to address this problem by looking at how *performance features of the SUT can be modeled* at an earlier stage using a model-based approach. With this research we strive to bridge the gap between model-based functional testing and performance testing using UML as the modeling language.

Our work investigates the applicability of model-based testing principles to performance testing. We plan to have a top-down approach; the process will go from requirements via test models to concrete test cases (see Figure 1). We will start by analyzing performance requirements and structure them in a requirements model. For the requirements modeling phase, we plan to look into how performance requirements explicitly can be formulated and grouped in terms of e.g., response times, memory usage, or bandwidth usage.

Further, we will investigate how *performance requirements* can be collected and quantified into acceptance criteria. This would facilitate the process of analyzing and

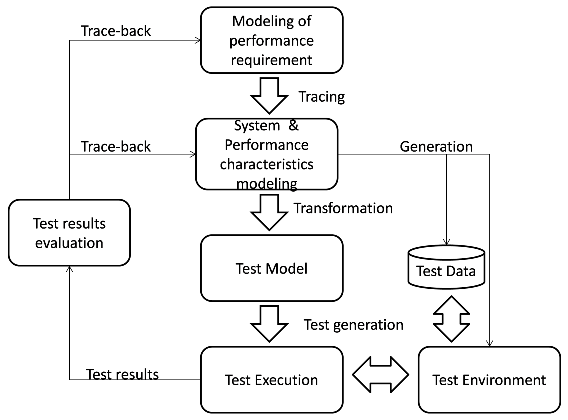

Figure 1. Model-based performance testing approach

understanding the results once they are available. When conducting performance testing, it is difficult to give a verdict if the test results are satisfactory. Hence, we plan to have a systematic approach of how performance requirements are measured and quantified in order to facilitate the process of analyzing performance test results. We plan to build *requirement models*, which will be used to capture, structure, and interrelate performance requirements. Exiting modeling languages as like the System Modeling Language (SysML) [22] or custom domain specific modeling languages (DSML) will be used for specifying the requirements.

The specifications of the system under test will be constructed from the requirements. The specifications, the *performance models*, will describe the SUT in terms of both functionality and performance characteristics. We will investigate how performance aspects of the system can be modeled using standardized UML profiles, like SysML, UTP, or MARTE. In addition, we plan to employ and adapt existing research done in the area of Software Performance Engineering for the application domain we are targeting.

Before using the performance models for test generation, the quality of the models has to be enforced. We plan to do this by defining an approach, in which the models are created based on specific modeling guidelines and checked both for inconsistencies with respect to performance requirements and for common modeling mistakes. Hence, we plan to have a structured way of validating the performance models and we will investigate the possibility of using languages like the Object Constraint Language (OCL) [23] for model validation.

Once the performance models are built, we intend to used them for the *generation of the test artifacts* used for performance testing. Besides generating functional tests based on selected traditional coverage criteria, we will look at how the test data (and its volume) used by the test scripts can be quantified, inferred, and generated from higher levels of the process. We will put special focus in data unicity, data distribution, and in generating invalid data. We will also investigate how the consistency of existing databases can be checked using the abstract models of the SUT, or how the test and load distributions per execution node can be derived.

We also plan to provide *traceability of performance requirements* across the entire process. More precisely, we plan to define an approach, in which the performance requirements are traced from models to generated test artifacts, taken into account during test execution, and included in the test reports. As one of the challenges of performance testing stands in analyzing the test reports, we will look into finding correlations between the resulting test reports and the specified performance requirements. We consider that such an approach will facilitate the interpretation of the test results and tracing of identified performance problems back to the performance models.

VI. Conclusion and Future Work

Although challenging, we believe that applying the principles of model-based testing to performance testing will bring benefits in terms of improved coverage, tool support, and automation. With this paper, we hope to trigger comments and interest in this topic.

References

[1] Compuware, "Applied performance management survey," Oct. 2006.

[2] E. Weyuker and F. Vokolos, "Experience with performance testing of software systems: issues, an approach, and case study," *Software Engineering, IEEE Transactions on*, vol. 26, no. 12, pp. 1147 –1156, dec 2000.

[3] C. U. Smith and L. G. Williams, *Performance Solutions*, ser. Object Technology. Addison-Wesley, 2002.

[4] Object Management Group, "Unified Modeling Language," http://www.omg.org/spec/UML/2.0/, last accessed May 22, 2010.

[5] M. Utting and B. Legeard, *Practical model-based testing: a tools approach*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2006.

[6] T. Keskinpala, "Model based performance testing of distributed large scale systems," Ph.D. dissertation, Vanderbilt University, 2009.

[7] M. Shams, D. Krishnamurthy, and B. Far, "A model-based approach for testing the performance of web applications," in *Proceedings of the 3rd international workshop on Software quality assurance*. ACM, 2006, p. 61.

[8] B. Pozin, R. Giniyatullin, I. Galakhov, and D. Vostrikov, "Model-based Technology of Automated Performance Testing," *SYRCoSE 2009*, p. 93, 2009.

[9] M. Fowler and K. Scott, *UML distilled: applying the standard object modeling language*. Addison-Wesley Reading, MA, 1997.

[10] P. Harmon and M. Watson, *Understanding UML: the developer's guide: with a Web-based application in Java*. Morgan Kaufmann, 1998.

[11] M. Fowler, *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003.

[12] P. Kähkipuro, "UML-based performance modeling framework for component-based distributed systems," *Performance Engineering*, pp. 167–184, 2001.

[13] D. Petriu, C. Shousha, and A. Jalnapurkar, "Architecture-based performance analysis applied to a telecommunication system," *IEEE Transactions on Software Engineering*, vol. 26, no. 11, pp. 1049–1065, 2000.

[14] J. Merseguer and J. Campos, "Software performance modeling using uml and petri nets," *Performance Tools and Applications to Networked Systems*, pp. 265–289.

[15] R. Mirandola and V. Cortellessa, "Uml based performance modeling of distributed systems," *UML 2000 The Unified Modeling Language*, pp. 178–193.

[16] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Transactions on Software Engineering*, pp. 295–310, 2004.

[17] Object Management Group, "UML Profile for Schedulability, Performance, and Time (STP)," http://www.omg.org/technology/documents/formal/schedulab ility.htm, last accessed May 22, 2010.

[18] ——, "UML Profile for Modeling and Analysis of Real-time and Embedded Systems (MARTE)," http://www.omg.org/spec/MARTE/1.0/, last accessed May 22, 2010.

[19] Object Management Group, "UML 2.0 Testing Profile," document formal/05-07-07, available at http://www.omg.org/, last accessed May 22, 2010.

[20] M. Woodside, G. Franks, and D. C. Petriu, "The future of software performance engineering," in *FOSE '07: 2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 171–187.

[21] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *FOSE '07: 2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 85–103.

[22] Object Management Group, "Systems Modeling Language Specification," http://www.omg.org/spec/SysML/1.1/, last accessed May 22, 2010.

[23] ——, "Object Constraint Language, may 2006," http://www.omg.org/spec/OCL/2.0/PDF/, last accessed May 22, 2010.