SOABench: Performance Evaluation of Service-Oriented Middleware Made Easy[*]

Domenico Bianculli[†]
Faculty of Informatics
University of Lugano
Lugano, Switzerland
domenico.bianculli@usi.ch

Walter Binder
Faculty of Informatics
University of Lugano
Lugano, Switzerland
walter.binder@usi.ch

Mauro Luigi Drago
DEEP-SE group - DEI
Politecnico di Milano
Milano, Italy
drago@elet.polimi.it

ABSTRACT

SOABench is a framework for the automatic generation, execution and analysis of testbeds for evaluating the performance of service-oriented middleware. Testbeds can be characterized in terms of the composite services to execute, the workload to generate, the deployment configuration to use, the performance metrics to gather, the data analyses to perform on them, and the reports to produce.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*testing tools*; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

Experimentation, Measurements, Performance

Keywords

Middleware, performance evaluation, web service compositions, testbed generation, experiment automation

1. INTRODUCTION

Service-oriented architectures (SOAs) are realized by deploying and executing several software services on the top of a dedicated middleware infrastructure, hereafter named *service-oriented middleware*, which offers different specific functionalities, such as service registration and discovery, assembly of composite services, service execution, and service management. Given its predominant role in the realization of an SOA, it is very important to assess the performance of its middleware components.

Evaluating the performance of distributed systems such as service-oriented systems consists in, at a minimum, executing a series of tests on different machines over different networks, possibly using several remote clients. Each test is characterized by a specific workload, and ends with the collection and the computation of some performance metrics (e.g., response time, throughput). When performed manually or with a limited amount of automation, this task can be cumbersome and error-prone. While there have been several proposals (e.g., Weevil [4]) for automating experimentation for distributed systems, most of the tools are not tailored to a specific domain and thus their adoption in the service-oriented computing domain becomes impracticable. Moreover, the tools (e.g., Genesis [3], MaramaMTE [2]) specific to the realm of SOAs mainly deal with the generation of testbeds for (Web) services and service compositions, and do not focus on the middleware components.

We addressed this problem in prior work [1], where we illustrated the architecture of a framework for automating experimentation with service-oriented middleware, and reported its application in a case study about benchmarking the performance of different BPEL engines.

We foresee the use of the framework for the evaluation and comparison of the performance of different service execution platforms (e.g., BPEL engines), and for assessing the impact of service-oriented middleware enhanced with different capabilities (e.g., monitoring, reputation management, self-healing, self-tuning), in various scenarios and under different workloads. Example questions that the framework could help answering are:

- *Which execution platform handles best the workload for given experiments on a given platform?*
- *What is the scalability of a certain execution platform, in terms of the maximum number of users and requests it can handle without showing failures?*
- *What is the overhead on the service execution due to a certain enhanced feature of the middleware (e.g., monitoring, self-tuning)?*
- *In which settings does the use of feature X within a middleware component outperform the use of feature Y within the same component?*

[*]This work has been partially supported by the EU under the grant agreement no. EU-FP7-215483-S-Cube and the IDEAS-ERC grant agreement no. 227977-SMScom; by the Swiss NSF under the grant agreements no. 125604 and no. 125337-CLAVOS, and the Sinergia project SOSOA, grant no. CRSI22_127386/1.

[†]This work was carried out while the author was an intern at Mission Critical Technologies, Inc., on site at NASA Ames Research Center.

These questions may be posed by researchers active in the service-oriented computing community, by enterprise ICT analysts, in the context of business decisions making regarding the implementation of SOAs in the enterprise, and by developers of service-oriented middleware components, to evaluate the performance and the scalability of their products under various workloads.

In this paper we give a brief overview of *SOABench*, the implementation of the framework described in [1], built with state-of-the-art technologies and targeting Web service-based SOAs, executing service compositions described as BPEL processes. *SOABench* is publicly available under the GPLv3 license at `http://code.google.com/p/soabench`.

2. SOABENCH AT A GLANCE

SOABench is composed of a testbed modeling environment and a tool chain, as depicted in Figure 1. The framework relies on well-established model-driven engineering techniques, such as model-to-model and model-to-text transformations, to design and generate the low-level artifacts required for executing an experiment.

The modeling environment includes a meta-model, which establishes the concepts necessary to define the tests to run and the testbed to use. This meta-model has been the basis for defining a Domain Specific Language (DSL), which can be used to design a testbed model for an SOA infrastructure. Such a model is characterized by the composite services to execute and the atomic services they invoke, the workload to generate, the deployment configuration to use, the performance metrics to gather, the data analyses to perform on them, and the reports to produce. Atomic services can be either references to the actual services, or mock-ups built using a mathematical description of the QoS attributes (e.g., response time, throughput). Moreover, a testbed model can include the definition of some events (e.g., the publication of a new service in a registry) that should occur during the execution of an experiment.

A testbed model is the input of the tool chain, which includes four main components. The first component is a set of *generator*s for creating executable versions of processes, the mock-ups of the external services they interact with, and the testing clients that create (concurrent) workload by invoking processes. A *test compiler* translates the testbed definition into a format understandable by the underlying platform we use for executing experiments. The *test driver* steers the experiment execution, and the *test analyzer* gathers and processes measurement data, producing as output reports including statistics computed from the measurements.

SOABench supports *technology independence* as a way to cope with the heterogeneity of SOA middleware. The framework guarantees that the testbeds it generates are reusable in as many settings as possible. This is achieved by treating SOA components as black-boxes characterized by a generic, common interface (e.g., the set of operations for deploying, starting and stopping BPEL processes). Moreover, platform dependencies are confined within a plug-in based mechanism. In addition, performance metrics are measured at the testbed infrastructure level, without the need for instrumenting the single components of the infrastructure by inserting profiling code.

The typical usage scenario of *SOABench* comprises several steps, which will be described below. All but the first step are performed automatically by the framework; indeed, the first one requires human intervention since it consists in the definition of the testbed model. The model is specified using the aforementioned DSL; the tools included in the modeling environment provide context assistance and consistency checking. Subsequently, the model is processed by the scripts provided with the framework, to generate the actual components of the testbed: the mock-ups of the services to be used, the testing clients, and the actual service compositions (e.g., BPEL processes) to execute. Additional artifacts generated during this phase include the deployment descriptors of the previously mentioned components, and helping scripts for starting/stopping them. For service mock-ups and testing clients, the framework also generates traces that mimic the non-functional behavior that each component will exhibit at run time. Pre-computing this information contributes to achieve *experiments repeatability*. Indeed, maximizing this goal is required to guarantee a fair comparison of different reference settings, and to mitigate uncontrollable factors in the measurement environment. However, given the complexity of the infrastructure, measurements are not exactly reproducible. To compensate for measurements variances, *SOABench* allows for repeating simulations several times, by using the traces of the behaviors to simulate.

The low-level artifacts generated in the previous step are then compiled to target the Weevil framework, which handles the actual execution of the experiment. After execution, *SOABench* automatically gathers all the execution logs, processes them, and creates a series of reports, which illustrate the results of the experiment with useful charts and statistics. For each specific performance metric, data collection and report generation are specified through a plug-in based mechanism.

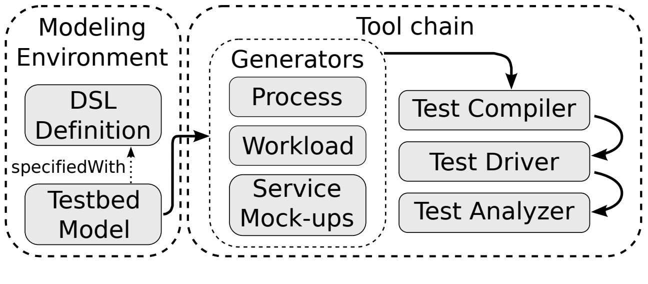

Figure 1: Architecture of *SOABench*

3. REFERENCES

[1] D. Bianculli, W. Binder, and M. L. Drago. Automated performance assessment for service-oriented middleware: a case study on BPEL engines. In *Proc. of WWW 2010*. ACM, April 2010.

[2] J. Grundy, J. Hosking, L. Li, and N. Liu. Performance engineering of service compositions. In *Proc. of SOSE'06*, pages 26–32. ACM, 2006.

[3] L. Juszczyk, H.-L. Truong, and S. Dustdar. GENESIS - a framework for automatic generation and steering of testbeds of complex web services. In *Proc. of ICECCS 2008*, pages 131–140. IEEE Computer Society, 2008.

[4] Y. Wang, M. J. Rutherford, A. Carzaniga, and A. L. Wolf. Automating experimentation on distributed testbeds. In *Proc. of ASE 2005*, pages 164–173. ACM, 2005.