

Model Driven Benchmark Generation for Web Services

Liming Zhu, Ian Gorton, Yan Liu

Empirical Software Engineering, National ICT
Australia

School of Computer Science and Engineering,
University of New South Wales

{Liming.Zhu, Ian.Gorton, Jenny.Liu@nicta.com.au}

Ngoc Bao Bui

Faculty of Information Technology, University of
Technology, Sydney, Australia

NgocBao.Bui@student.uts.edu.au

ABSTRACT

Web services solutions are being increasingly adopted in enterprise systems. However, ensuring the quality of service of Web services applications remains a costly and complicated performance engineering task. Some of the new challenges include limited controls over consumers of a service, unforeseeable operational scenarios and vastly different XML payloads. These challenges make existing manual performance analysis and benchmarking methods difficult to use effectively. This paper describes an approach for generating customized benchmark suites for Web services applications from a software architecture description following a Model Driven Architecture (MDA) approach. We have provided a performance-tailored version of the UML 2.0 Testing Profile so architects can model a flexible and reusable load testing architecture, including test data, in a standards compatible way. We extended our MDABench [27] tool to provide a Web service performance testing “cartridge” associated with the tailored testing profile. A load testing suite and automatic performance measurement infrastructure are generated using the new cartridge. Best practices in Web service testing are embodied in the cartridge and inherited by the generated code. This greatly reduces the effort needed for Web service performance benchmarking while being fully MDA compatible. We illustrate the approach using a case study on the Apache Axis platform.

Categories and Subject Descriptors

D.2.10 [Software]: Software Engineering – *Design*; D.2.11 [Software]: Software Engineering – *Software Architectures*; D.2.2 [Software]: Software Engineering – *Design Tools and Techniques*

General Terms

Design, Theory

Keywords

MDA; model-driven development; Performance; Testing; Code Generation; Web Service; Service-Oriented Architecture

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IW-SOSE'06, May 27–28, 2006, Shanghai, China.

Copyright 2006 ACM 1-59593-085-X/06/0005...\$5.00..

1. Introduction

Web services technologies have proven useful in the construction of enterprise-scale systems. However, many challenges remain, especially ensuring that Web services solution can meet specified performance requirements [2].

Various performance analysis models with prediction capabilities exist to evaluate architecture designs during early phases of the application development cycle [4] [8]. Applying them to Web services has shown promising results [16]. Utilizing these models requires two distinct activities be carried out by the application architect. The first requires the development of specific analytical models based on the application design. The second must obtain parameter values for a performance model using measurements or simulation. Both these activities require significant additional effort and specific expertise in performance engineering methods. Hence, they are key inhibitors that have prevented performance engineering techniques from achieving wide-spread adoption in practice [4].

With the growing interest in Model Driven Architecture (MDA)[19] technologies, attempts to integrate performance analysis with MDA and UML have been made, aiming to reduce the performance modeling effort required. Recent work has attempted model transformation from UML design models to method-specific performance analysis models [21]. Parameter values in these models also depend greatly on the underlying Web service framework used to implement the application. One method to obtain and tune these parameters is to run a benchmark application on the framework. This approach has proven to be useful with component-based technologies [11, 12] [17] and Web services [16]. Running benchmark applications can also help in predicting and diagnosing performance problems, including identifying bottlenecks, preliminary profiling and exploring core application characteristics.

An effective benchmark suite includes a core benchmark application, a load testing suite and performance monitoring utilities. There are existing industry benchmark standards and suites applicable to Web services (e.g. TPC-W v2 [9]), but these are not broadly suitable for performance modeling and prediction for a number of reasons. First, they are mainly designed for vendors to showcase and improve their products, rather than reflecting a specific application’s performance characteristics. The application logic in these benchmarks is

fixed and impossible to adapt to assist in predicting performance for a specific application under design. Second, these benchmark suites tend to be expensive to acquire and complex to use.

On the other hand, implementing a custom benchmark suite from scratch for Web services is costly and error-prone. This is due to the complexity of Web service frameworks and the higher than usual performance testing demands for Web services.

In our approach, we tackle the problem from two perspectives:

1) A benchmark implementation usually requires a large amount of container and framework infrastructure-related plumbing, even for a relatively simple benchmark design. Interestingly, this characteristic is particularly amenable to MDA-based code generation, which is efficient at generating repetitive but complicated infrastructure code. However, one capability that current MDA code generation frameworks lack is that they do not provide solutions to the generation of a load testing suite and performance data collecting utilities.

2) Load testing suites are often constructed in a bottom-up fashion. Manually produced code is coupled with test data and the System Under Test (SUT). Their structures are often not optimized for reuse and adaptation to a large number of load test scenarios and test data. In order to address this, we raise the level of load testing suite design by adopting and tailoring the UML 2.0 Testing profile [23]. The use of the profile encourages modular design of the load testing suite. Meanwhile, using MDA-based code generation, a large amount of reusable performance testing and data capture infrastructure code are also generated using “cartridges”. A cartridge is a collection of meta-model definitions, code generation handlers and templates.

The aim of our work is to automate the generation of complete Web services benchmark suites from a design description. The input is a UML-based set of design diagrams for the benchmark application, along with a load testing suite modeled in the UML 2.0 Testing Profile. The output is a deployable benchmark suite including the core benchmark application, load testing suite, and performance data collecting utilities.

To demonstrate the approach, the Apache Axis Web service platform [3] has been selected for our case study. The work is based on our previous research [27] on model driven customized benchmark generation and the MDABench toolkit [28]. This paper illustrates how to model the load testing suite using UML along with the core application design and generate deployable customized benchmark applications for Web Services platforms. Executing the generated benchmark application produces performance data in an analysis friendly format, along with automatically generated performance graphs.

This approach has a number of benefits:

1) The generated benchmark suite is based on a design that closely corresponds to the application of interest, and hence it captures the unique characteristics of the application. This should lead to the benchmark producing more representative measures of the eventual application.

2) Model driven code generation hides the complexities of the benchmark implementation from architects, and helps them focus on analyzing the benchmark results that are automatically produced. It also integrates best practices in Web service performance testing into the generation “cartridges” and the associated framework.

3) Following MDA standards, including the UML 2.0 Testing Profile, and using existing open source MDA frameworks significantly reduces the learning curve of the approach. It also takes advantage of existing code generation “cartridges” exploiting the latest technologies and platforms. The wide range of interoperable UML modeling tools (due to the MDA/UML compatibility standard) also makes the approach more amenable to adoption in practice.

4) Most importantly, the load testing data is modeled in a modular fashion with test data decoupled from the testing logic. This allows large numbers of testing scenarios to be accommodated and managed from a high-level model.

The next section discusses related work on Web service performance testing and MDA. We then introduce our approach in section 3. The case study based on Apache Axis is presented in section 4. We briefly discuss the reusability and extensibility in section 5 then conclude in section 6.

2. Related Work

2.1 Web Service Performance Testing

The Web service testing domain itself is relatively new. Most of the current work focuses on functional testing [6]. Performance-related Web service testing has focused on comparing different SOAP implementations [18] rather than application-specific performance. However, such application-specific performance is vital in situations like service level driven management [7] and QoS-aware Web services [5].

However, conducting Web service performance testing has its unique challenges:

- Web services are often consumed by external parties that are not controlled by the service provider. There are potentially high numbers of possible usage scenarios to be tested. Some of them are unpredictable and need to be performance tested on demand. This requires a flexible and reusable performance testing architecture.
- Web service operations are often coarse grained. Performance varies significantly, based on different XML payload sizes, when the same service is invoked. This demands the service not only be load tested using different numbers of concurrent users, but also under different payload sizes. These all increase the complexity of benchmarking.
- Due to the use of XML and overheads incurred by the interoperability standards of Web services, applications are more performance sensitive than traditional component-based technologies. Increasingly wider adoption of Web services in enterprise applications demand thorough performance testing before the system enters production.

Our approach directly addresses these problems by encouraging flexible load testing designs using the UML 2.0 testing profile

and providing supporting infrastructures through model driven code generation.

2.2 Model Driven Architecture

It has been argued that the MDA approach is a suitable for facilitating performance analysis of enterprise systems [22]. This usually involves deriving performance analysis models through model transformation [20, 24, 25] [13].

Analytical models can not work without data to populate the model parameters. For complicated commercial systems, measurements in the form of benchmarking [16] and prototyping [15] are used to obtain valuable information for architects. However, comprehensive benchmarking and prototyping can be very time consuming. As stated earlier, further exacerbating the problem, multiple benchmarks need to be executed for different performance scenarios such as different XML payloads and request mixes for different platforms.

We argue that deriving a customized benchmark application using MDA improves productivity and quality of the benchmark generation. Our approach in fact complements MDA based analytical model construction.

3. Model Driven Benchmark Generation for Web Services

Some pioneering work has been done on generating benchmark and prototyping applications using models, as in [14, 15] for component-based distributed applications. However, it has several limitations in terms of standards compatibility, leveraging existing code generation “cartridges” and load test modeling [27]. Thus, we have developed MDABench [28] using AndroMDA [1] for customized benchmark generation. AndroMDA is an open source MDA code generation framework with a large amount of server-side technology cartridges. MDABench is essentially a client-side cartridge for benchmark generation. In this work, we extended MDABench to further provide a Web services benchmark generation cartridge.

3.1 Core Web Service Application Generation

The core application generation simply involves modeling the application in UML and exploiting existing Web service code generation cartridges. After modeling, all necessary source files including business method interfaces and implementation skeletons are generated. Implementation logic needs to be manually provided to produce a deployable application. Modeling and generating the benchmark application is not therefore a distinct engineering step from normal development activities though the amount of development resources required differ. It can be considered as one of the early steps in an incremental development process instead of a throw away performance prototyping activity. One Platform Independent Model (PIM) can also be used to generate different deployable applications for different Web service platforms with little modification. This can greatly reduce the cost and consequently the hurdle of performing performance engineering in practice. More details can be found at [27].

3.2 Load Test Suite Generation

First, we model the load testing behavior using the UML 2.0 Testing Profile [23]. This profile is an OMG standard, representing a comprehensive superset of existing widely used testing frameworks such as JUnit. Our new Web service cartridge extends our existing J2EE performance testing cartridge. To this end, we have implemented the following stereotypes in the UML 2.0 Testing Profile: *SUT* (System under Test), *Test Context*, *Test Component*, *Data Pool*, *Data Partition and Test Cases*. *Data pool*, *data partition* and *test case* can be used for modeling different test data for the same or different test cases. Different types of test data can be modeled and associated with test cases under different circumstances. Please see [27] for more details.

In the process of developing our approach and implementation, we integrated performance engineering best practices from the various leading technologies and our own experience.

3.2.1 Test Suite Internal Design

The internal design of the load testing infrastructure is inspired by ECperf/SPECjAppser [9]. Clients, Controllers and Drivers are separate components which coordinate the testing in a flexible distributed manner. The Driver interprets the run properties, communicates with Clients and configures them according to the model. Clients register themselves with a controller, and the controller aggregates the final performance results. Through this modular design, the Controller/Driver can be reused across applications. Clients will be generated according to Web service consuming logic and the modeled load testing data. The Controller/Driver infrastructure can evolve independently from the generated Clients. This design style has become a de-facto standard in distributed testing community.

3.2.2 Performance Metrics and Collection Utilities

Performance metrics are based on industry standards such as TPC-W v2 [23]. Some example metrics data for Web service include:

- **SIRT (Web Service Interaction Response Time)**: the time taken to perform a successful web interaction
- **SIPS (Service Interactions Per Second)**: the average number of SIPS completed during a measurement interval. We also use transactions per second (TPS) to refer to per second service interaction.

In this manner, analysis friendly performance data can be collected and standard benchmark reports can be generated if appropriate report generation templates are developed.

We also included more metrics such as timing details and distribution statistics. Distribution statistics allows a more in-depth view of the performance results compared to average response time and throughput. These enable us to identify critical irregularities and their causes during our test runs of the benchmarks. We also provide capabilities to store the timing details. Timing details capture the time to execute each individual operation to be recorded in the results repository. This may of course incur performance and storage overhead. However, the timing information allows further correlation with other internal or external events which may have significant

performance impacts. Performance collection utilities are based on our own extensive experience on performance testing.

3.2.3 Load Testing Configuration

In addition to basic load testing configuration support [27] such as concurrent users and request mixes, we have also provided more realistic incremental and spike test simulation. They are based on Grinder 3 [10] and Microsoft Visual Studio 2005. Some examples are:

config.initialProcesses: The initial number of processes to start with.

config.processIncrement: The number of processes to increase or decrease for an incremental time interval.

config.processIncrementInterval: The time interval between starting up or stopping new processes.

config.stabilizationperiod: An estimated time before a steady state period is reached.

All these best practices are essentially encapsulated in the cartridge, and its use automatically supports these best practices through generated code structures and utilities. This is one of the main motivations behind our approach.

We provide a complete template for generating a default implementation for all operations on SUT with randomly generated data based on a data pool model. A database seeder is also generated to repopulate the database before a new test.

These capabilities greatly reduce the extra effort involved in using the suite in load testing activities, in which performance testing is the main interests of the software engineer.

4. Case Study

We reuse the Stock-Online J2EE model from our previous benchmark generation work to demonstrate the minimal effort required for conducting Web service performance testing using the same model. The Stock-Online system is a proven benchmark for evaluating various middleware platforms. The original system was developed for different J2EE platforms. Due to platform differences, there was significant effort involved in implementing the same design for different platforms, and keeping the benchmark application in line with component technology advancements required significant ongoing effort.

4.1 Benchmark application modeling for Web services

The server side logic is modeled using the UML and AndromDA profiles. In Figure 1, class *Broker* represents the entry point to the full server side UML model, which is not shown here. It is marked with the stereotypes `<<WebService>>`, `<<Service>>` and `<<SUT>>`. One of the motivations behind MDABench is to reuse the same model as much as possible and hide platform differences in the cartridge. When the `<<WebService>>` stereotype is used, all the operations can be accessed as Web services. If only selected methods need to be exposed as Web services, a method level stereotype `<<WebServiceOperation>>` can be used.

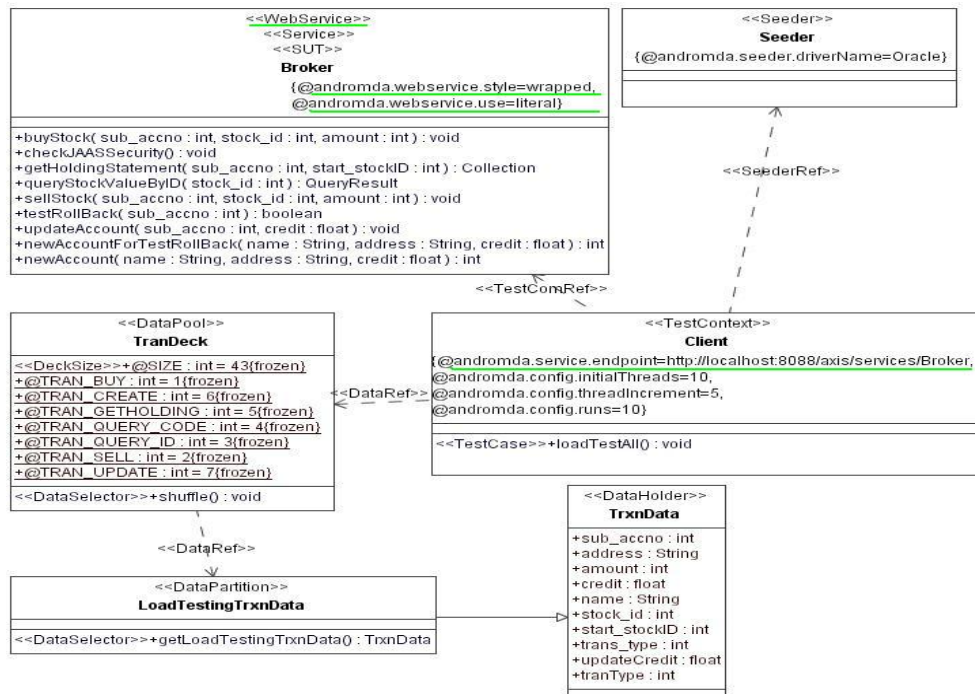


Figure 1. Benchmark Modeling for the Axis Web service platform

In Web service modeling, a number of performance parameters can be set through configuration on the model directly. These include WSDL binding styles (RPC/Document) and binding use styles (Encoded/Literal). As we have annotated on the diagram through tagged values on the *Broker* class, we use the doc/literal wrapped pattern. The wrapped pattern is a slightly improved variation of the commonly used doc/literal style. This is considered the best configuration for performance. Such tuning largely depends on the server side cartridge. However, the values can be queried by the load testing client to conduct necessary style-specific testing and performance measurement.

The client side modeling conforms to the UML 2.0 testing profile. Client is the <<TestContext>> which consists of test cases. Only the default *loadTestAll()* test case is included with its default implementation generated. For simplicity, all the test data is modeled in *TrxnData* from which <<DataPartition>> *LoadTestingTrxnData* is derived. In more complicated

situations, several test data classes may exist for different XML payloads. They can be associated with <<TestContext>> through <<DataRef>> for different load test scenarios at different times. In <<DataPool>> *TranDeck*, we can also indicate the transaction mix percentage as tagged values.

Performance testing settings such as concurrent workload, incremental simulation and test step durations are configured through tagged values included in the <<TextContext>> stereotype.

There is little change on the client side modeling when compared to a J2EE model except the configuration of the endpoint through a tagged value. All the extra changes involved are encapsulated in the client cartridge. If a Web service targeted model is detected, the cartridge will generate Web service specific look-ups and a Web service client while the rest of the testing logic and data is untouched.

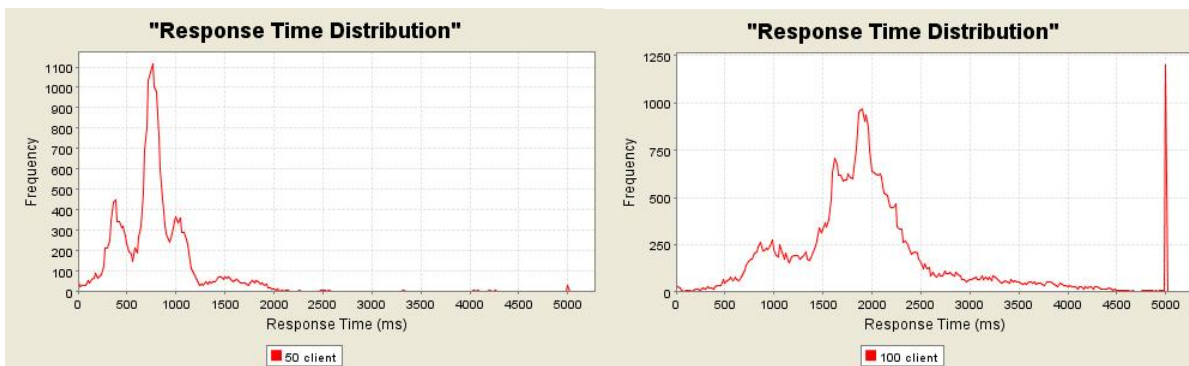


Figure 2. Samples of response time distribution for 50/100 clients on Axis

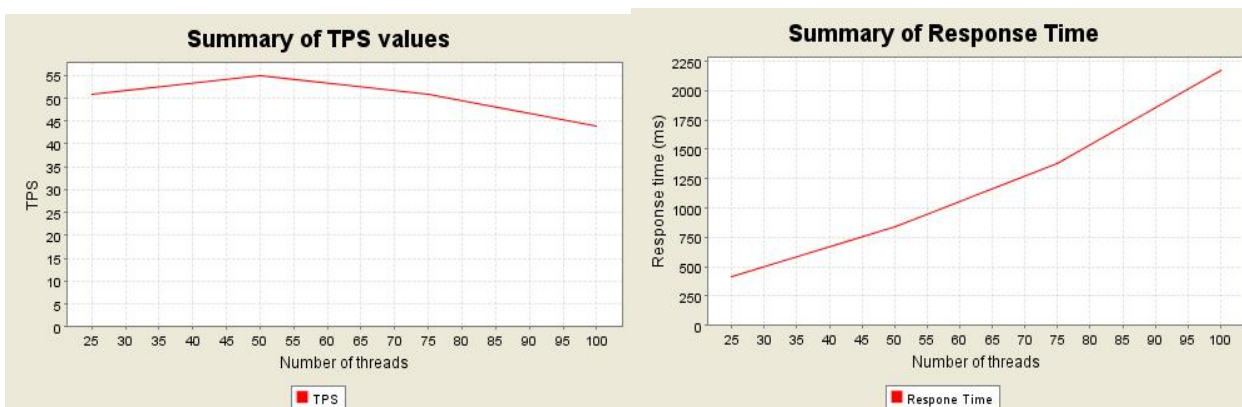


Figure 3. Average response time and throughput on Axis in an incremental requests simulation

4.2 Performance Output for Axis Web Service Platform

By running MDABench, Web service directory structures are generated for the Apache Axis platform. These consist of a

MDA directory for storing the exported UML model, and directories for storing source code and the future deployable application. Project property files for specifying dependencies on the targeted platforms and other deployment configurations

are also generated. We then copy the exported UML model into the designated MDA directory and run the code generation engine. Source code is generated based on the UML model. For the client side, the complete load testing suite is generated without the need for further modification. The load testing logic and random test data is derived from the load testing UML model and method signatures of the Web service interface.

In terms of the Web service implementation, business logic inside each Web service method needs to be manually added by placing implementation code into a separate directory. This prevents overriding manual modification by subsequent code generation iterations. After adding the server side business logic code, MDABench generates the deployable package.

Deploying and running the Web service benchmark produces the follow results:

- Figure 2 shows the response time distribution for 50 and 100 clients respectively. The spikes at the end of the 5000ms indicate all response times longer than 5000ms. Average response time and detailed performance data are stored in analysis friendly logs.
- Figure 3 shows the average response time and TPS in terms of number of threads in an incremental requests simulation scenario.

Performance analysis of these data is beyond the scope this paper.

5. Reusability and Extensibility

5.1 Model Reusability

When we first developed MDABench, we focused our attention on the J2EE platform. This implementation adhered to three design principles:

- Decouple the test suite from the server side technology. Most information required to connect to the server side is gathered through querying the server-side Platform Independent Model (PIM).
- We choose not to instrument any server-side technology cartridges. This allows the server-side cartridge to be evolved separately and be kept up-to-date to the latest technology developments.
- The load testing suite modeling capability is strictly divided into a platform independent profile and platform dependent markings. The platform independent profile is a combination of the UML 2.0 Testing Profile and some load testing domain specific languages. The platform dependent markings are all tagged values.

These design principles have not only allowed us to move MDABench to Web service platform but also achieved a number of benefits for users, especially ones who have investment in existing models. There are virtually no changes of the load testing suite architecture model except the configuration of the endpoint through a tagged value (See Figure 1) when we move from an existing J2EE load testing model to a Web service load testing model. After the initial once-off effort of developing the Web services cartridge, one

student took one day to change the benchmark model to the Web services annotation and conduct the test successfully.

5.2 MDABench Extensibility

MDABench can be easily extended. There are a number of ways developers can extend it:

- Major utility components of the MDABench provide either interfaces or abstract classes for overwriting existing implementations.
- Components to interpret modeling elements strictly follow a chain of command pattern to enable delegations to any new model transformation and code generation interpreters.
- A templating capability within cartridges provides a simple extension mechanism.

Using these mechanisms, extending MDABench from J2EE to Web services took us relatively little effort. However, it will be difficult to directly extend MDABench to the Microsoft .Net platform. Though MDABench theoretically could generate .Net applications, the UML-based modeling environment is not encouraged in Microsoft Visual Studio development environment. Microsoft has launched its own initiative on model driven development (Software Factories) and uses the Domain Specific Language (DSL) as its modeling language. Thus, we are currently developing an MDABench equivalent using DSL. It will take advantage of the existing Visual Studio testing capabilities but raise the level of testing into a model driven level.

6. Conclusion and Future Work

This paper has presented an approach to generate a customized benchmark application from architecture designs for Web service platform using MDA.

A benchmark design is modeled with platform independent models in UML. A corresponding load testing suite is modeled following a subset of the UML 2.0 Testing Profile. Deployable code is then generated for both the core benchmark design and its associated load testing suite. The load test suite generator has been developed by the authors, and fully integrates with the core application generation. A case study using the Axis Web services platform for the Stock-Online benchmark suite have demonstrated the tools and the generated outputs from load tests.

This approach has several advantages over proprietary model-based CASE tool environments for benchmark generation. Using MDA and exiting open source MDA frameworks reduces the learning curve and training effort required, and improves model traceability and tool interoperability. The default implementation and test data generation saves a large amount of load testing effort considering the high demand of Web service testing.

There are still limitations of this approach. The default implementation of the load testing suite is still relatively simple. It covers only successful test scenario generation and does not produce more interesting stress testing data. Currently, users have to implement such scenarios or produce higher quality stress testing data manually. However, auto generation of such data has been successfully applied in other

areas [26]. We are considering integrating these methods both at the modeling level and in the default implementation in future versions. We are also working on linking MDABench's modeling ability with current Web service quality of service standards such as Web Service Level Agreement to reflect requirements modeling needs.

7. Acknowledgments

National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

8. References

- [1] AndroMDA, "AndroMDA v3.0M3", <http://andromda.org/>.
- [2] M. Aoyama, S. Weerawarana, H. Maruyama, C. Szyperski, K. Sullivan, and D. Lea, "Web Services Engineering: Promises and Challenges," in Proceedings of the International Conference on Software Engineering (ICSE), 2002.
- [3] Apache, "Apache Axis 1.3 Final", <http://ws.apache.org/axis/>.
- [4] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: a survey," *IEEE Transactions on Software Engineering*, vol. 30 (5), pp. 295-310, 2004.
- [5] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "QoS-Aware Replanning of Composite Web Services," in Proceedings of the International Conference on Web Services, 2005.
- [6] W. K. Chan, S. C. Cheung, and K. P. H. Leung, "Towards a Metamorphic Testing Methodology for Service-Oriented Software Applications," in Proceedings of the First International Workshop on Services Engineering (SEIW), 2005.
- [7] D. D. Dan, R. Kearney, R. K. A. Keller, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef, "Web services on demand: WSLA-driven automated management," *IBM Systems Journal*, vol. 43 (1), pp. 136-155, 2004.
- [8] G. Denaro, A. Polini, and W. Emmerich, "Early Performance Testing of Distributed Software Applications," in Proceedings of the ACM International Workshop on Software Performance (WOSP), 2004.
- [9] ECperf, "ECperf v1.1", <http://java.sun.com/j2ee/ecperf/index.jsp>.
- [10] P. Gomez, P. Aston, and etc, "The Grinder V3.0-beta23", <http://grinder.sourceforge.net/>.
- [11] I. Gorton and A. Liu, "Evaluating the performance of EJB components," *IEEE Internet Computing*, vol. 7 (3), pp. 18-23, 2003.
- [12] I. Gorton, A. Liu, and P. Brebner, "Rigorous evaluation of COTS middleware technology," *IEEE Computer*, vol. 36 (3), pp. 50-55, 2003.
- [13] V. Grassi and R. Mirandola, "A Model-driven Approach to Predictive Non Functional Analysis of Component-based Systems," in Proceedings of the UML 2004 workshop on Models for Non-functional Aspects of Component-Based Software, 2004.
- [14] J. Grundy, Y. Cai, and A. Liu, "Generation of distributed system test-beds from high-level software architecture descriptions," in Proceedings of the 16th Annual International Conference on Automated Software Engineering (ASE), 2001.
- [15] J. Grundy, Z. Wei, R. Nicolescu, and Y. Cai, "An environment for automated performance evaluation of J2EE and ASP.NET thin-client architectures," in Proceedings of the Australian Software Engineering Conference (ASWEC), 2004.
- [16] Y. Liu and I. Gorton, "An Empirical Evaluation of Architectural Alternatives for J2EE and Web Services," in Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC), 2004.
- [17] Y. Liu, A. Fekete, and I. Gorton, "Design-Level Performance Prediction of Component-Based Applications," *IEEE Transactions on Software Engineering*, vol. 31 (11), pp. 928-941, 2005.
- [18] A. Ng, S. Chen, and P. Greenfield, "An evaluation of contemporary commercial SOAP implementations," in Proceedings of the Fifth Australasian Workshop on Software and System Architectures, Melbourne, 2004.
- [19] OMG, "Model Driven Architecture", <http://www.omg.org/mda/>.
- [20] M. J. Rutherford and A. L. Wolf, "Integrating a Performance Analysis Kit into Model-Driven Development," in Proceedings of the the 5th GPCE Young Researchers Workshop 2003, Erfurt, Germany, 2003.
- [21] J. Skene and W. Emmerich, "Model Driven Performance Analysis of Enterprise Information Systems," *Electronic Notes in Theoretical Computer Science*, vol. 82 (6), pp. 1-11, 2003.
- [22] J. Skene and W. Emmerich, "A model-driven approach to non-functional analysis of software architectures," in Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE), 2003.
- [23] TPC, "TPC Benchmark W (TPC-W)", <http://www.tpc.org/tpcw/spec/TPCWV2.pdf>.
- [24] T. Weis, A. Ulbrich, K. Geihs, and C. Becker, "Quality of service in middleware and applications: a model-driven approach," in Proceedings of the Eighth IEEE International Enterprise Distributed Object Computing Conference (EDOC), 2004.
- [25] C. Yilmaz, A. M. Memon, A. A. Porter, A. S. Krishna, D. C. Schmidt, A. Gokhale, and B. Natarajan, "Preserving distributed systems critical properties: a model-driven approach," *Software, IEEE*, vol. 21 (6), pp. 32-40, 2004.
- [26] J. Zhang and S. C. Cheung, "Automated test case generation for the stress testing of multimedia systems," *Softw. Pract. Exper.*, vol. 32 (15), pp. 1411-1435, 2002.
- [27] L. Zhu, J. Liu, I. Gorton, and N. B. Bui, "Customized Benchmark Generation Using MDA," in Proceedings of the 5th Working IEEE /IFIP Conference on Software Architecture, 2005.
- [28] L. Zhu, J. Liu, I. Gorton, and N. B. Bui, "MDAbench: A Tool for Customized Benchmark Generation Using MDA," in Proceedings of the Object-Oriented Programming, Systems, Languages and Applications (OOPSLA), 2005.