Benchmark generation using domain specific modeling

Author/Contributor:
Bui, Bao; Zhu, Liming; Gorton, Ian; Liu, Yan

Publication details:
2007 Australian software engineering conference ASWEC 07, Proceedings
pp. 169-180
9780769527789 (ISBN)
1530-0803 (ISSN)

Event details:
2007 Australian software engineering conference ASWEC 07
Melbourne, Australia

Publication Date:
2007

DOI:
https://doi.org/10.26190/unsworks/400

License:
https://creativecommons.org/licenses/by-nc-nd/3.0/au/
Link to license to see what you are allowed to do with this resource.

Downloaded from http://hdl.handle.net/1959.4/38550 in https://unsworks.unsw.edu.au on 2023-12-02

Benchmark Generation using Domain Specific Modeling

Ngoc Bao Bui[1], Liming Zhu[2], Ian Gorton[3], Yan Liu[2]
[1]*School of Computer Science and Engineering, University of New South Wales, Australia*
[2]*Empirical Software Engineering, National ICT Australia Ltd, Australia*
[3]*Pacific Northwest National Laboratory, USA*
{betty.bui,liming.zhu,yan.liu}@nicta.com.au; ian.gorton@pnl.gov

Abstract

Performance benchmarks are domain specific applications that are specialized to a certain set of technologies and platforms. The development of a benchmark application requires mapping the performance specific domain concepts to an implementation and producing complex technology and platform specific code. Domain Specific Modeling (DSM) promises to bridge the gap between application domains and implementations by allowing designers to specify solutions in domain-specific abstractions and semantics through Domain Specific Languages (DSL). This allows generation of a final implementation automatically from high level models. The modeling and task automation benefits obtained from this approach usually justify the upfront cost involved. This paper employs a DSM based approach to invent a new DSL, DSLBench, for benchmark generation. DSLBench and its associated code generation facilities allow the design and generation of a completely deployable benchmark application for performance testing from a high level model. DSLBench is implemented using Microsoft Domain Specific Language toolkit. It is integrated with the Visual Studio 2005 Team Suite as a plug-in to provide extra modeling capabilities for performance testing. We illustrate the approach using a case study based on .Net and C#.

1. Introduction

To conduct business effectively over the Internet, organizations require high-performance and scalable distributed systems that are able to process thousands of client requests per seconds. Component based technologies such as CORBA, EJB and .NET provide mechanisms and services to facilitate the construction of such enterprise scale systems. However, it is challenging to ensure that such systems meet performance requirements, especially early in the development life cycle when only design diagrams are available.

A number of design based performance prediction models have been proposed to solve this problem [1]. These approaches involve two main activities: developing analytical performance models and collecting performance parameter values to populate the model. Among the techniques proposed to obtain parameter values, benchmarking has proved useful as it provides quantitative measures of the performance characteristics of middleware components, which have a significant impact on such systems [2, 3].

The results from industry benchmark standards are not adequate for thorough benchmarking, as their results mainly show the performance characteristics of vendor specific platform products. For example, ECperf [4], a popular benchmark application, is designed specifically for J2EE application servers. The results released from ECperf are specific to highly optimized vendor platforms and hardware configurations rather than a specific business application of interest. The construction of a highly customized benchmark application is hence desirable. However this activity is both time and cost consuming because it requires complex, error prone and repetitive middleware oriented development and deployment. To further exacerbate the problem, multiple benchmarking applications may need to be constructed for different target platforms for comparison or deployment considerations. Therefore it is necessary to devise techniques and tools to facilitate the efficient construction of benchmark applications on multiple platforms.

Domain Specific Modeling (DSM) is a promising approach to fulfill this requirement. DSM is one of the main approaches used for Model Driven Development (MDD). The basic idea of DSM is to model software

Proceedings of the 2007 Australian Software Engineering Conference (ASWEC'07)
0-7695-2778-7/07 $20.00 © 2007 IEEE

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: The University of New South Wales. Downloaded on November 26, 2008 at 22:40 from IEEE Xplore. Restrictions apply.

products using Domain Specific Languages (DSL) and produce implementations automatically from the high level models. DSL is designed to support a specific domain and to solve a particular and identifiable problem through domain-specific semantics and abstractions [5]. It defines a set of domain specific modeling constructs, rules and constraints extracted from a problem domain. Each construct usually has a visual representation that can be used to build a domain model in a visually configurable manner. The model represents the problem at a high level without involving any implementation aspects. Domain specific code generators then interpret these models and produce the implementation for a specific platform and programming language. Full code generation is possible as the descriptions in the model and the generator capture all the required aspects of the application [6].

DSM raises the level of abstraction significantly. By using a DSL for modeling, designers can work directly with problem domain concepts when specifying the solution. This can potentially provide a number of benefits [7-9] including:

- User friendly visual notations for the domain
- Domain specific analysis and optimization through the model
- Task automation through code generation
- Product line adoption for a specific domain,
- Facilitate data description and system configuration

We consider benchmarking a unique area that can benefit from DSM due to:

- Benchmark applications are usually required for different platforms with different configurations. DSM can facilitate system configuration and platform specific code generation.
- The work involved in benchmarking for different platforms and different testing scenarios is highly repetitive. The task automation benefits provided by DSM are consequently potentially helpful.
- A large number of performance testing scenarios require ease of description of test case data. Using DSM for *data modeling* provides major improvements over the current in-code and file-based test data provision.

This paper describes a DSM based benchmark generation approach. The main contribution of this paper is that we propose a DSL, DSLBench, for automated benchmark generation in the domain of benchmarking-based performance evaluation. DSLBench includes modeling constructs extracted from the performance testing domain.

We have implemented a supporting DSM environment. It allows developers to model and generate the load testing part of a benchmark application suite. With the support of code generators associated with DSLBench, the final benchmark application for a specific platform can generated and deployed immediately from the domain models specified in DSLBench. The approach has a number of benefits:

- It improves productivity significantly through task automation and high level system configuration.
- It improves communication through a new visual language for benchmark modeling.
- It improves the flexibility of benchmark applications through code generators for different platforms and test case data modeling capabilities.

Currently, DSLBench is implemented using the Microsoft Domain Specific Language toolkit. It is integrated with the Visual Studio 2005 Team Suite as a plug-in. A C# code generator is provided.

The paper is structured as follows. The development process for a DSM environment is introduced in section 2. Section 3 applies the general process in the benchmarking domain and Section 4 provides a case study using this DSM environment. Then, an evaluation of the method is presented in Section 5 and is followed by a discussion of model based benchmark generation in Section 6. Section 7 concludes the paper.

2. DSM environment and development

To fully utilize the benefits of DSM, a DSM environment for a specific domain needs to be built. The environment includes a new DSL, a model editor for the DSL, code generators including templates for mapping models to platform specific implementations and an optional domain specific framework containing reusable domain services to be called by the templates.

However, developing such a DSM environment from scratch requires significant engineering effort. A practical solution is to use DSL meta-tools to create a DSM environment that can meet the requirements of a specific domain. Such meta-tools allow language designers to define new modeling languages with a set of modeling constructs. It also provides facilities to build model editors and domain specific code generators.

Currently, there are a number of DSL tools [10] ranging from research tools to commercial environments. Some of them, such as MetaEdit+, are standalone tools while others, such as Microsoft DSL toolkit, are integrated into IDEs. Tight IDE integration provides better access to infrastructure services within an IDE's ecosystem and better bi-directional engineering. However, it limits the tool to be effectively used outside the IDE.

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: The University of New South Wales. Downloaded on November 26, 2008 at 22:40 from IEEE Xplore. Restrictions apply.

From available DSL tools, we have chosen the Microsoft DSL toolkit [11] for developing a DSL and associated DSM environment for the benchmark generation domain. The Microsoft DSL toolkit is a part of the Microsoft Visual Studio 2005 SDK. With this tool, the user can specify a domain model and create a custom graphical designer to build the modeling constructs of a DSL. It also provides a template based code generation engine that enables DSL users to create their own generators in addition to the ones provided by the DSL designers.

3. DSM based Benchmark Generation

3.1. Benchmarking domain

Benchmarking is the process of running a specific program or workload on a specific machine or system, and measuring the resulting performance. Generally, a benchmark suite for a component based application includes three parts [2]:

- A core benchmark application which consists of the main application logic. This will exercise essential platform services.
- A load testing suite which will stress test the benchmark application and record the relevant measurement results.
- Configuration files used for declaratively customizing load testing configurations.
- A measurement reporting utility for representing the measurements of performance metrics.

The core benchmark application logic may be available already, written manually or generated from models. Ideally, the core benchmark application can be modeled using a general purpose modeling language, such as UML or a DSL created for a particular domain. Then we can exploit the model integration between the core benchmark models and load testing models in a unified modeling environment. In the case that there is no existing code generation framework for the domain of the core benchmark application, the code generator has to be developed for that domain. This is out of the scope of this paper. This paper focuses only on building a DSM environment for modeling and generating the load test suite.

A load test suite emulates the workload of the core benchmark application. It also collects performance measurements and generates necessary reports. Generally, a load test suite handles the following:

- Key scenarios and test cases: performance-critical scenarios of the benchmark application and test cases for those scenarios
- Metrics: the performance metrics to be collected when executing load tests
- Work load and load simulation: the total load among the various usage scenarios and load generation to simulate the load for each test case
- Resource utilization threshold: The resource utilization threshold is the amount of resources consumed by the application at peak load levels. For example, the processor overhead of the application should not be more than 10 percent.

A DSM environment has been built to support the modeling and code generation of a load test suite for the benchmark application. This environment includes two main parts: a performance modeling language named DSLBench and a code generator that produces C# code for load test suite executed in Visual Studio 2005 Team Suite. The environment reuses Visual Studio's load testing infrastructure and integrates with it seamlessly.

The rest of this section gives a description of those components and explains how a DSM environment can support modeling and generation of a load test suite.

3.2. DSLBench

The development of DSLBench was carried out using the following process:

- Decision phase: identify the purpose of the new DSL. We identified four purposes 1) provide a new visual language for benchmark generation 2) use automation for repetitive and error-prone tasks 3) enable high level system configuration 4) offer test case data modeling capabilities
- Analysis Phase: This step involves identifying the domain concepts in the benchmarking domain, their logical relationships as well as associated constraints. As we will show later, such domain concepts and constraints come from existing testing standards and load testing frameworks.
- Design Phase: This step involves using the DSL meta-tool to create language constructs for the above identified domain concepts. This includes both textual and visual language constructs within the limits of the Microsoft DSL toolkit. A meta model for DSLBench is thus created.
- Implementation Phase. The implementation includes code generator construction. This step usually defines code generation templates that map elements in the models into implementation artifacts.

The identified concepts should be generic and self-contained. They should ensure that the load test suite modeled and generated based on these concepts can cover the functionalities required for performance testing listed in section 3.1. These concepts are

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: The University of New South Wales. Downloaded on November 26, 2008 at 22:40 from IEEE Xplore. Restrictions apply.

abstracted from resources that cover the state-of-the-art of load testing utilities, i.e.:

- Testing architecture proposed in the OMG UML 2.0 Testing Profile [12]
- ECperf
- Grinder 3 [13]
- Visual Studio load testing infrastructure
- Entities specified in MDA based benchmark generation framework [18]

The main concepts and associated rules are as follows:

- **Coordinator:** A Coordinator can combine and connect various elements of a load test suite. It realizes key scenarios and generates corresponding test cases with suitable input values. A Coordinator can be connected to one data generator for random data generation and test components that specified the system under the test. A coordinator includes several attributes
  - *IsConstantPattern:* Specify whether usage pattern is constant or not. A constant usage pattern does not change during the load test. In contrast, a step load pattern is used to specify a user load that increases with time up to a defined maximum user load.
  - *DatabaseConnectionString:* Specify the location of the back-end database. This attribute helps to provide the default implementation for all test cases that need to access to a back-end database as well as enable automatic database population during the performance testing. This concept is obtained from the ECperf infrastructure.
- **DataGenerator:** the component that is responsible for generating random values for input into test cases and populating the database. The DataGenerator concept originates from the testing architecture in the OMG 2.0 Testing Profile.
- **TestComponent:** Specify the server side component under test with a set of scenarios that significantly affect the performance of the component. A test component can have many methods. Similar to DataGenerator, it is taken from the OMG test architecture.
- **Method:** A method represents a performance critical scenario. Information on a method provides details to generate test cases for the key scenarios. By default, one key scenario has one test case. Each method has a *TestMix* attribute that specifies the probability of a virtual user running a given test in a load test scenario. With this value, the simulated load becomes more realistic.
- **Threshold:** Thresholds are used to plan for resource utilization. A threshold is a rule that is set on an individual performance counter to monitor system resource usage during performance testing. One coordinator can have many thresholds since many threshold rules can be set during one running instance of load test suite. The concept of threshold and its relationships with the rest of the constructs comes from Visual Studio load testing infrastructure.
- **DatabaseTable:** A DatabaseTable corresponds to a table in the backend database for seeding test data.

The Microsoft DSL toolkit provides a general DSM environment that can be customized to specify the underlying domain model of a DSL. In the domain model, domain concepts are represented by classes and relationships with different value properties.

Figure 1 shows the domain model of the performance testing specified in the domain model designer tool provided by the Microsoft DSL toolkit.

Concepts and rules are visually represented with graphic notations so that designers of a DSL can interact with the language. The definition of graphical notations can be done with Visual Studio DSL tool support that provides a way to map the performance concepts to graphical notations used in a load test model. For example, the Visual Studio DSL tool allows users to define their own graphic notation with shapes and colors they prefer. The Visual Studio environment makes it easy to create, read, and maintain domain model.

An individual concept can be represented as standalone constructs or associated with other constructs depending on the concept's properties, and its relationships to others. For example, a *Coordinator* is the main component that contains properties and has relationships to other components, so it should be represented as an independent construct. A *Threshold* is associated with a *Coordinator*, and therefore it is modeled as an attribute of a *Coordinator*.

3.3. The code generator

Without a code generator, a DSL only helps to model and document the solution for better communication and thus has limited value for software development. A code generator is essential to support a self-contained DSL. Domain specific code generators extract and interpret domain concepts and rules specified in the model and map them to specific types of output such as source code, text files or XML based configuration files. It is possible to build domain specific code generators that support full code generation, as far as the template for generating platform specific code is available.

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: The University of New South Wales. Downloaded on November 26, 2008 at 22:40 from IEEE Xplore. Restrictions apply.

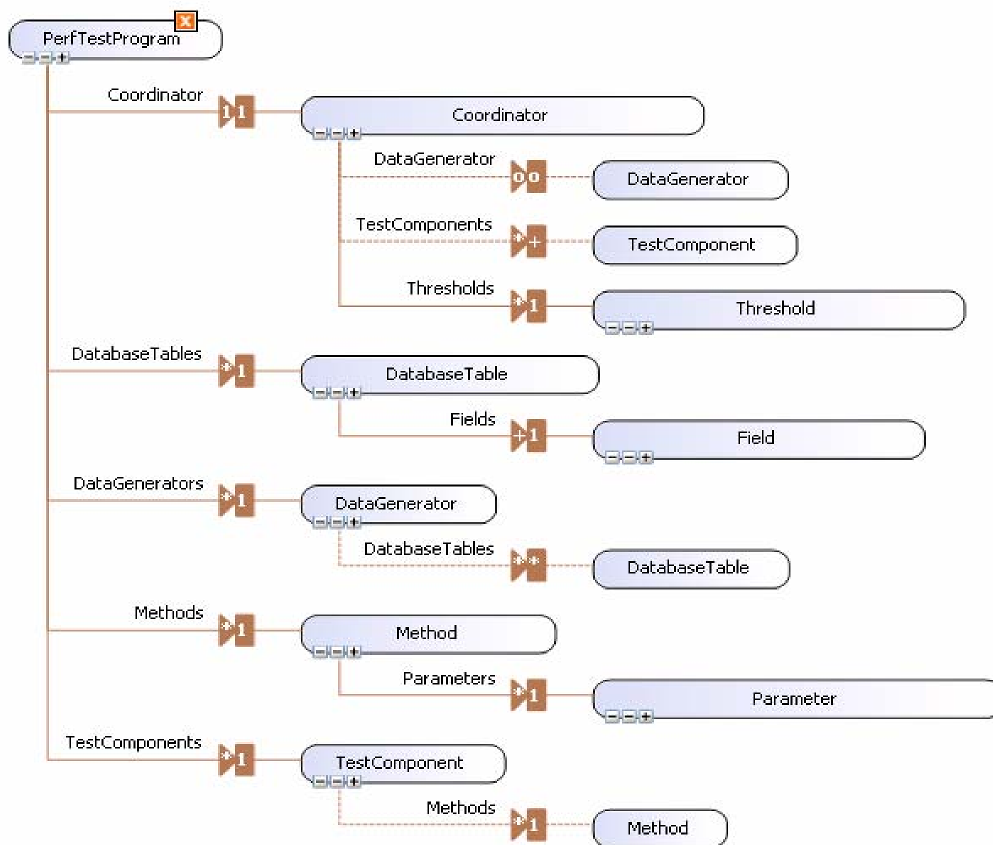
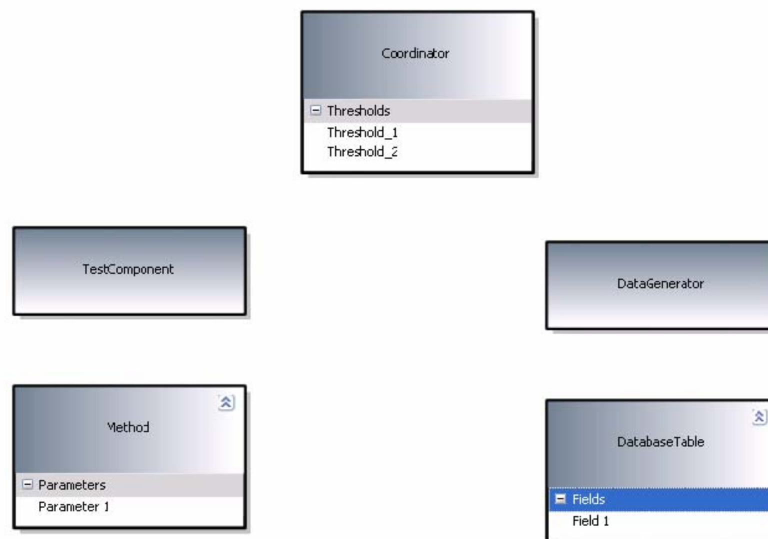


Figure 1: The performance testing domain model



IEEE COMPUTER SOCIETY

Authorized licensed use limited to: The University of New South Wales. Downloaded on November 26, 2008 at 22:40 from IEEE Xplore. Restrictions apply.

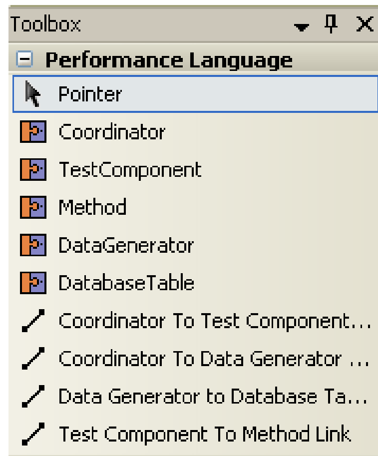


Figure 2: Some modelling constructs of performance domain specific language, DSLBench and their appearance in the editor.

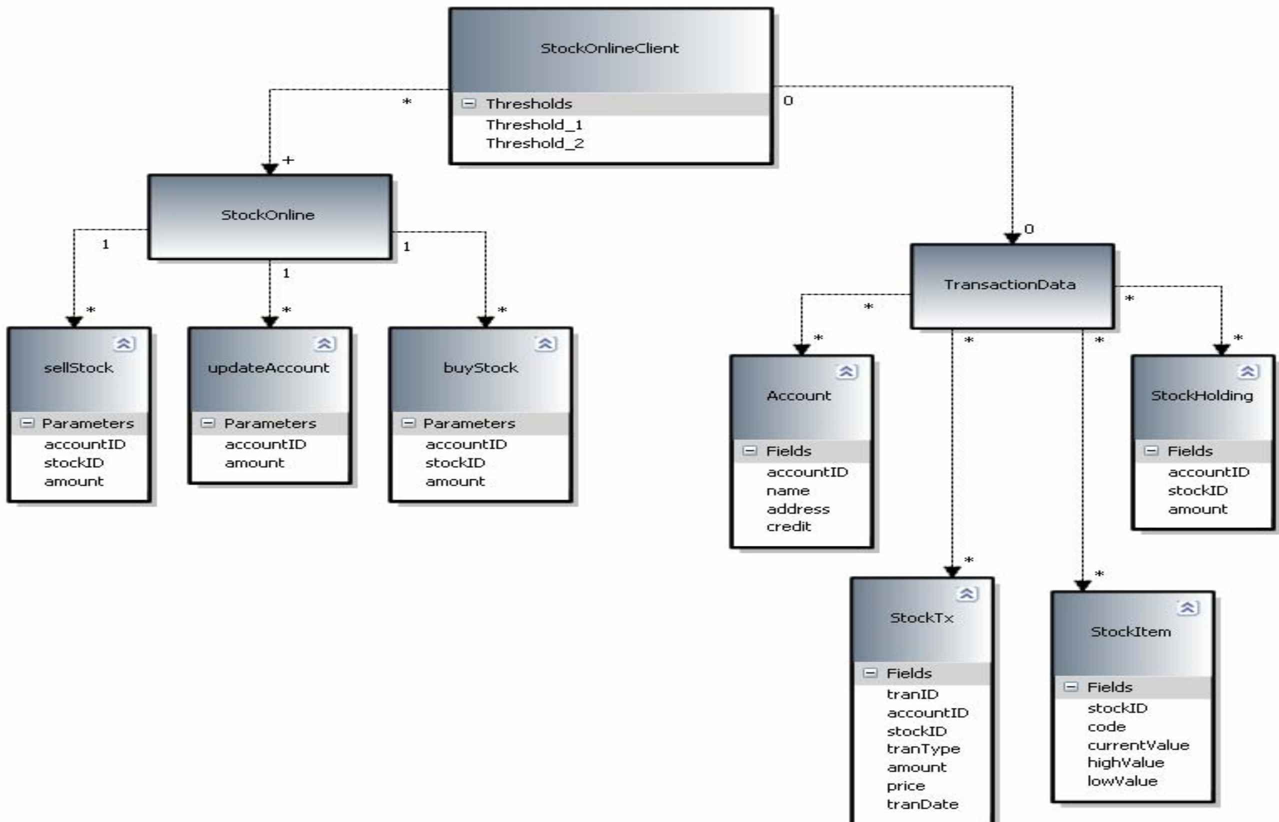


Figure 3: The load test model of Stock-Online application specified in DSLBench

The Visual Studio DSL tool includes a text template transformation tool that allows defining template-based code generators. A code generator takes the DSL definition of the domain model based on a domain model, which is imported in XML format, and produces code that implements each concept element as output in C# classes. The code generators also validate the domain model based on the constraints specified in the domain model, and raise errors and warnings accordingly. The Visual Studio DSL tool integrates the .Net environment with other windows-based performance monitoring and testing components. The current version of generated load test suite can seamlessly utilize the functionality of these components [14].

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: The University of New South Wales. Downloaded on November 26, 2008 at 22:40 from IEEE Xplore. Restrictions apply.

3.4. Integration with Visual Studio

The DSM environment integrates with Microsoft Visual Studio seamlessly in a number of ways. Firstly, the DSL toolkit produces a distribution of the DSL and its model editor (designer). The distribution can be installed and deployed in Visual Studio as a plug-in.

Second, the current load testing facilities in Visual Studio rely on a wizard, scripts and configuration files. The DSLBench environment provides a modeling capability for these scripts and configuration files. The files generated by DSLBench are compatible with the XML file format used by Visual Studio load testing. The synchronization is bi-directional. Any changes in the model will be reflected in VS load testing configuration and any changes in the VS load testing configuration will be reflected in the performance testing model.

Finally, all the code generated is part of the VS environment, which taps into the facilities provided by the IDE.

4. Case study

Stock-Online [15] is used as case study to illustrate the DSM based benchmark application generation approach. *Stock-Online,* a synthetic stock trading system has some typical stock trading tasks such as buying and selling stock, and creating user account details. The application requires a database to store the account and stock data, and this database is preloaded before each test with a known initial data set. As tests are run, clients execute a test script in which a known mix of transactions is requested. However, each client randomizes the order in which the transactions are executed so that locks on data are also randomized.

The load testing suite for Stock-Online is modeled using the modeling constructs of DSLBench. The load test model consists of three sections: test components, back-end database, and coordinating component or coordinator. *ClientDemo* is the *Coordinator* that controls the load test process. *ClientDemo* contains various load test specific information that is exported into the load test file deployed by the load test tool and a configuration file specific to the load test suite. Those files allow the testers to declaratively customize load test configuration. Also, all changes in those files can be reflected in the model to prevent hand-coded configuration information being overwritten.

The load test suite includes a *DataGenerator* named *TransactionData* and a number of *DatabaseTable* elements that model the back-end database. Each *DataTable* has several *Fields* with specific data types and range of values. The information helps to produce valid data to populate the database before each test. The system under test is modelled by *TestComponent* and associated *Methods* elements; in this case, *Stock-Online* is the *TestComponent* with some of typical transaction operations such as *sellStock*, *buyStock,* and *updateAccount* each of which has a specific value for *TestMix* to represent the transaction mix. The *TestComponent* Input data for each operation are specified in *Parameters* with specific types and range of values. In case a *Parameter* has an equivalent value stored in the database, the input value for it will be synchronized with the stored value. For each *Field* and *Parameter,* it is necessary to specify data type and value limits to ensure valid data to be produced. In the final implementation of load suite, there is a generated class that implements simple logic to generate values of primitive types such as integer, double or string within specified limits. The complete model of the load test suite for *Stock-Online* application is presented in Figure 3.

The complete and deployable load test suite is generated including test cases in C# code and configuration files. All files are exported to a testing project in Visual Studio 2005, compiled and then executed by the load test tool. The load test suite executes on the top of the load test tool which provides facility to display, record and store various performance metrics during and after the testing as shown figure 4. All of the testing results are stored in SQL server and can be extracted later. Also, a summary of the testing, including response time and throughput are produced to provide the testers an overview of the results.

5. Evaluation

The performance specific modeling language, DSLBench, helps developers to model a load test suite and produce deployable code. The modeling process is easy and straightforward and does not require much knowledge of performance testing, as DSLBench covers concepts and rules abstracted from the performance testing domain and hides all the complexity of the performance testing process.

The domain model is defined in modular fashion. The model is conceptually divided into three sub-models, namely persistent data for modeling back-end databases, test components for modeling functionality to be tested, and a coordinating component for modeling performance specific features such as load patterns. The modularization feature ensures the extensibility of the language, so that each sub-model of DSLBench can be reused in defining a new performance testing suite. The addition of new concepts can be done through the extension of sub-

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: The University of New South Wales. Downloaded on November 26, 2008 at 22:40 from IEEE Xplore. Restrictions apply.

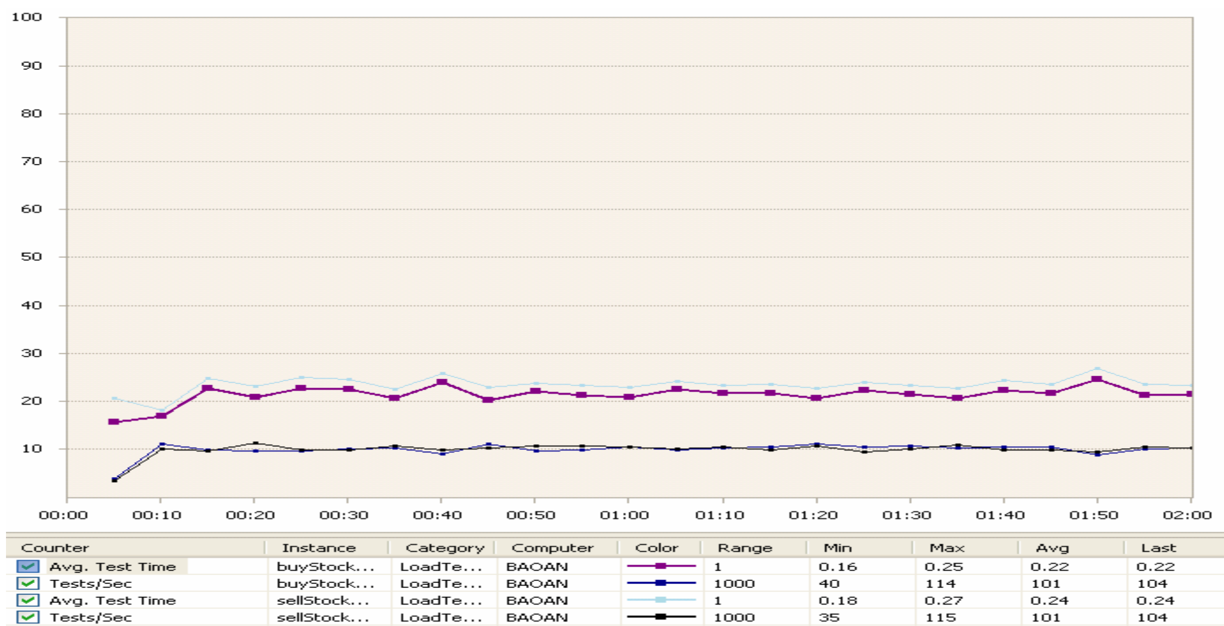
models and does not affect the fundamental structure of the language between sub-models.

The current code generator supports the .NET platform and is built on the top of the testing tool of Visual Studio 2005 team suite to utilize its features including load simulation, visualization of results and results storage. The code generation template is customizable which simplifies the construction of the code generator for a domain model of a specific domain. The integration of the Microsoft DSL toolkit into Visual Studio facilitates the installation and deployment of DSLBench inside Visual Studio. However, due to the restriction of the Microsoft DSL toolkit used for the language development, it does not support compilation and execution of the generated source code in the same project. The source code needs to be moved to another Visual Studio project for execution. This restriction of the tool limits the usability and maintainability of DSL since the generated code has to be moved manually or by scripts to another project every time the model is changed and code is re-generated.

Task Automation

It took considerable effort to develop the original Stock-Online application for different middleware platforms, conduct load testing and collect performance data. Extending it to .NET environment and Web service platform later involved another round of similar effort. Though familiarization of the business logic might make it easier, the repetitive and error prone parts for testing and infrastructure plumbing are the components that have to be rebuilt for each platform.

Using our DSLBench, one student, the main developer of the toolkit but with limited experience with load testing in Visual Studio and .NET, took 2 days to model the system and conduct the load tests. We realize that this effort comparison is anecdotal, and we do not have accurate effort data from the original development team. It is also easier to develop the same application the second time. However, it is preliminary evidence that the effort spent on the basic plumbing code and load test suite has been absorbed by cartridge developers and code generation. Hence we believe the productivity savings afforded by task automation through DSM are potentially considerable.



Counter	Instance	Category	Computer	Color	Range	Min	Max	Avg	Last
Avg. Test Time	buyStock...	LoadTe...	BAOAN		1	0.16	0.25	0.22	0.22
Tests/Sec	buyStock...	LoadTe...	BAOAN		1000	40	114	101	104
Avg. Test Time	sellStock...	LoadTe...	BAOAN		1	0.18	0.27	0.24	0.24
Tests/Sec	sellStock...	LoadTe...	BAOAN		1000	35	115	101	104

Figure 4: Graph showing test results produced by the test tool of Visual Studio 2005 Team Suite

Data Modeling

Existing performance tools have scripting and test recording features, but they lack test data modeling capabilities and are extremely limited in features for test data generation. According to our past experience, gaining full access to any real data and workload profile for measurement is often difficult in the real world. To represent the system as realistically as possible and reduce testing effort, DSLBench's test data modeling and generation produces high quality

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: The University of New South Wales. Downloaded on November 26, 2008 at 22:40 from IEEE Xplore. Restrictions apply.

data for a large number of requests types and their combinations. However, the default data generation is still relatively simple. It covers only successful testing scenario generation. In real applications, performance of exception handling and transaction rollback is also a major concern. Currently, users have to produce test case data for such scenarios through customized data generation templates. We are considering integrating these both at the modeling level and in the default implementation in future versions.

Visual Language

DSLBench provides extra visual modeling capabilities for performance testing. The effectiveness of the visual language was only validated through practical use by a few researchers. We are planning to conduct a thorough empirical investigation on the following aspects:

- Visualization improvements evaluation comparing to UML
- Evaluation of language expressiveness using a systematic approach

6. Related Work

Model Driven Architecture (MDA) is another major approach towards model driven development. MDA proposes a way to describe an application using a set of models specified in UML at different levels of abstraction to produce the final product via model transformations. Similar to DSM, MDA supports code generation from architecture designs directly. Some pioneering work has been done on generating benchmark and prototyping applications using models, as in [17-19]. In particular, [17] introduces *SoftArch/MTE,* a tool to generate deployable test-beds for distributed systems from high level specifications, automatically deploy and send testing results back to those models. However, these approaches have several limitations. The code generators for the chosen technologies are built from scratch by the researchers. They do not utilize any extensible generator frameworks, or draw upon the large pool of existing code generation "cartridges" for the latest technologies that are maintained by an active community. When code generation cartridges are not exploited, any change to the chosen target technology or the introduction of a new technology requires significant extra work from the researchers. DSLBench exploits Visual Studio's load testing infrastructure by integrating with existing load testing facilities.

An MDA based benchmark generation framework to generate benchmark suite for J2EE application [20] was developed based on an open source MDA framework, AndroMDA [21]. The framework makes uses platform-specific UML profiles and code generators to generate the core benchmark application. A load test profile and code generator are created and integrated into the framework to support modeling and full code generation for the load test. The MDA based framework has proved to greatly reduce the effort and expertise need for benchmarking with complex component technology [20]. However, some of the performance testing concepts such as meta-data for the test data generations are not covered by standard UML profiles, as these are usually very domain specific. Tailoring UML with domain specific concepts can be inefficient in terms of capturing the semantics of a domain model. Although extra mechanisms such as the Object Constraint Language (OCL) could be used along with UML to express additional semantics, they are limited to UML constructs. This may incur problems of model accuracy and comprehension.

The DSM based approach proposed in this paper complements the MDA project above to fully exploit the potential of DSM.

7. Conclusion

This paper presents a customized benchmark generation approach based on DSM, which currently supports the .NET platform. A DSM environment has been built, which makes it possible to model a load test suite for component based systems using domain abstractions and to generate the complete, deployable code from the models.

The DSM approach has some advantages over other benchmark generation frameworks. It offers a simple modeling environment that reduces the learning curve since it directly uses concepts abstracted from the performance testing domain. In addition, it creates a complete load test suite that is able to perform comprehensive load testing including data population, random data generation and handling test cases and performance value collection.

8. Acknowledgement

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

9. References

[1] Balsamo, S., et al., *Model-Based Performance Prediction in Software Development: a Survey.*

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: The University of New South Wales. Downloaded on November 26, 2008 at 22:40 from IEEE Xplore. Restrictions apply.

Software Engneering, IEEE Transactions, 2004. **30**(5): p. 16.

[2] Gorton, I., A. Liu, and P. Brebner, *Rigorous Evaluation of COTS middleware technology*. Computer, 2003. **36**(3): p. 6.

[3] Gorton, I. and A. Liu, *Evaluating the Performance of EJB Components*. IEEE Internet Computing, 2003. **7**(3): p. 6.

[4] *ECperf*. [cited; Available from: http://jcp.org/aboutJava/communityprocess/final/jsr004/index.html.

[5] Mernik, M., J. Heering, and A.M. Sloane, *When and how to develop domain-specific languages*. ACM Computing Surveys, 2005. **37**(4): p. 29.

[6] Iseger, M., *Domain-specific modeling for generative software development*, in *ITArchitects*. 2005.

[7] Kelly, S. *Improving Developer productivity with Domain Specific Modelling Languages* 2005 [cited 2006 7 September]; Available from: http://www.developerdotstar.com/mag/articles/domain_modeling_language.html.

[8] Pohjonen, R. and S. Kelly. *Improving productivity and time to market*. 2002 [cited; Available from: http://www.metacase.com/papers/DrDobbs_Domain-Specific_Modeling.html.

[9] Tolvanen, J.-P. *Domain Specific Modeling: Welcome to the Next Generation of Software Modeling*. 2005 [cited 2006 7 September]; Available from: http://www.devx.com/enterprise/Article/29619.

[10] *DSL tools*. [cited; Available from: www.dsmforum.org/tools.html.

[11] AndroMDA. *AndroMDA v3.0M3*. 2005 [cited; Available from: http://andromda.org/.

[12] *UML 2.0 Testing Profile Specification*. [cited; Available from: http://www.omg.org/cgi-bin/doc?ptc/2004-04-02.

[13] *The Grinder*. [cited; Available from: http://grinder.sourceforge.net/.

[14] *Visual Studio 2005 Team Edition for Software Testers*. [cited; Available from: http://msdn.microsoft.com/vstudio/teamsystem/products/test/default.aspx.

[15] Gorton, I., *Enterprise Transaction Processing Systems: Putting the CORBA OTS, Encina++ and Orbix OTM to Work*. 2000: Addison-Wesley.

[16] *Apache JMeter*. [cited; Available from: http://jakarta.apache.org/jmeter/usermanual/index.html.

[17] Grundy, J., Y. Cai, and A. Liu, *SoftArch/MTE: Generating Distributed System Test-beds from High-level Software Architecture Descriptions*. Automated Software Engineering, 2005. **12**(1): p. 34.

[18] Rutherford, M.J. and A.L. Wolf. *A Case for Test-Code Generation in Model-Driven Systems*. in *The 2nd International Conference on Generative Programming and Component Engineering*. 2003. Erfurt, Germany.

[19] Grundy, J., et al. *An environment for automated performance evaluation of J2EE and ASP.NET thin-client architectures*. in *Australian Software Engineering Conference (ASWEC)*. 2004.

[20] Zhu, L., et al. *Customized Benchmark Generation Using MDA*. in *the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*. 2005.

[21] *AndroMDA*. [cited; Available from: www.andromda.org.

Authorized licensed use limited to: The University of New South Wales. Downloaded on November 26, 2008 at 22:40 from IEEE Xplore. Restrictions apply.