MDAbench: A Tool for Customized Benchmark Generation Using MDA

Liming Zhu
School of Computer Science and Engineering, University of NSW, Australia
Liming.Zhu@nicta.com.au

Yan Liu
Empirical Software Engineering Program, National ICT Australia
Jenny.Liu@nicta.com.au

Ian Gorton
Empirical Software Engineering Program, National ICT Australia
Ian.Gorton@nicta.com.au

Ngoc Bao Bui
Faculty of Information Technology, University of Technology Sydney, Australia
NgocBao.Bui@student.uts.edu.au

ABSTRACT

Designing component-based application that meets performance requirements remains a challenging problem, and usually requires a prototype to be constructed to benchmark performance. Building a custom benchmark suite is however costly and tedious. This demonstration illustrates an approach for generating customized component-based benchmark applications using a Model Driven Architecture (MDA) approach. All the platform related plumbing and basic performance testing routines are encapsulated in MDA generation "cartridges" along with default implementations of testing logic. We will show how to use a tailored version of the UML 2.0 Testing Profile to model a customized load testing client. The performance configuration (such as transaction mix and spiking simulations) can also be modeled using the UML model. Executing the generated deployable code will collect the performance testing data automatically. The tool implementation is based on a widely used open source MDA framework AndroMDA. We extended it by providing a cartridge for a performance testing tailored version of the UML 2.0 Testing Profile. Essentially, we use OO-based meta-modeling in designing and implementing a lightweight performance testing domain specific language with supporting infrastructure on top of the existing UML testing standard.

Categories and Subject Descriptors

D.2.11 [**Software**]: Software Engineering – *Software Architectures*; D.2.2 [**Software**]: Software Engineering – *Design Tools and Techniques*; D.3.4 [**Programming Languages**]: Processor – *code generation*

General Terms

Design, Language, Theory

Keywords

MDA, model-driven development, CASE Tools, Performance, Testing, Code Generation

1. INTRODUCTION

With the growing interest in Model Driven Architecture (MDA) [5]technologies, attempts to integrate performance analysis with MDA and UML have been made, aiming to reduce the performance modeling effort required in component-based software development. Recent work has focused on model transformation from UML design models to method-specific performance analysis models, using standard UML performance profiles. To obtain model parameter values, a number of different methods can be used. One is to use a benchmark application. This approach has proven to be useful [3] with component--based technologies. An effective benchmark suite includes a core benchmark application, a load testing suite and performance monitoring utilities. But implementing a custom benchmark suite for a component platform from scratch is also costly and tedious. A benchmark implementation usually requires a large amount of container and 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 generation of a load testing suite with data collection infrastructure for a given benchmark.

The aim of this tool is to automate the generation of complete 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 client modeled in a performance tailored version of the UML 2.0 Testing Profile [6]. The output is a deployable complete benchmark suite including monitoring/profiling utilities. The target platform is server-side component platforms, currently the J2EE component framework. Executing the generated benchmark application produces performance data in an analysis friendly format, along with automatically generated performance graphs.

2. CUSTOMIZED BENCHMARK GENERATION USING MDA

The tool is built on top of an open source extensible framework - AndroMDA [4], for MDA based code generation. We extended AndroMDA to support a performance testing tailored version of the UML 2.0 Testing Profile for load test suite generation. Benchmark designers can model their own benchmark application along with a load testing suite in UML. The overall structure of the benchmark generation and related process workflow is presented in the boxed area in Figure 1.

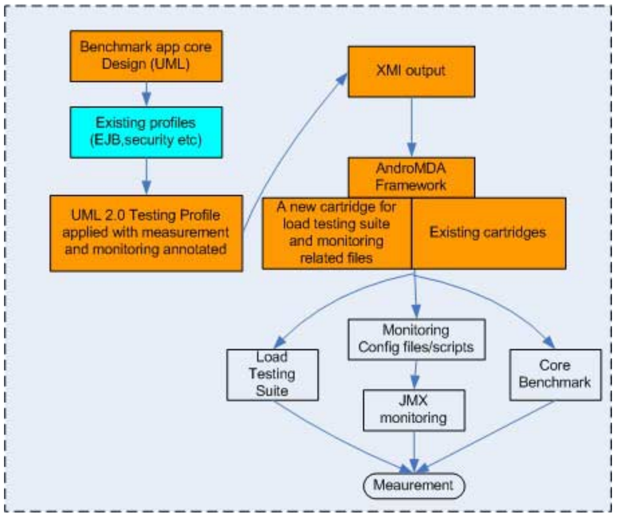
Benchmark app core Design (UML)
Existing profiles (EJB,security etc)
UML 2.0 Testing Profile applied with measurement and monitoring annotated
XMI output
AndroMDA Framework
A new cartridge for load testing suite and monitoring related files
Existing cartridges
Load Testing Suite
Monitoring Config files/scripts
JMX monitoring
Core Benchmark
Meaurement

Figure 1. Model Driven Benchmark Suite Generation

The core benchmark application generation simply exploits MDA development techniques and existing cartridges using AndroMDA. The load testing behavior is modeled using a tailored version of the UML 2.0 Testing profile [6]. 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, Test Case. Each stereotype includes a set of tagged values for various purposes including correlations to the SUT, test data modeling, test scenario modeling (such as transaction mix) and performance testing configurations (number of runs, ramp up time and etc.).

The AndroMDA extension for load test modeling and generation results in a new cartridge. The design and implementation of the cartridge combines OO-based meta-modelling and domain specific language design. We also provide a complete template for generating a default implementation of the loadTestAll() test case with randomly generated data based on a data pool model and transaction mix. A database seeder is also generated to populate the database. A simple client side example modeled using this profile in MagicDraw 8.0 is shown in Figure 2. Executing the generated deployable code will collect the performance testing data automatically in analysis friendly format and graphs.

3. RELATED WORK

Some pioneering work has been done on generating benchmark and prototyping applications using models, as in [1, 2]. However, these have several limitations:1) The code generators for the chosen technologies are built from scratch by the researchers. They do not draw upon the vast pool of exiting code generation “cartridges” for latest technologies. Any change to the chosen target technology or the introduction of a new technology requires significant extra work. 2) These methods do not follow the MDA standards. Existing industry experience on UML and code generation is therefore not leveraged. 3) The load testing part of the benchmark suite can not be comprehensively modeled compared to using the UML 2.0 testing profile. The latter makes load test suite modeling more modular, reusable and modifiable. Our approach directly addresses these limitations.

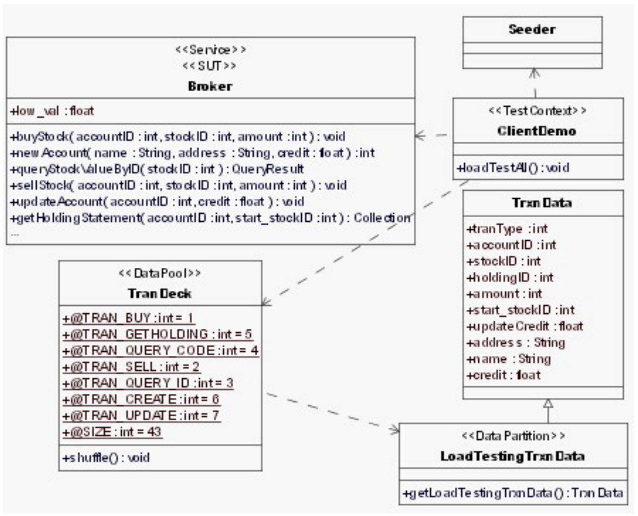
<<Service>>
<<SUT>>
Broker
+low_val : float
+buyStock(accountID : int, stockID : int, amount : int) : void
+newAccount(name : String, address : String, credit : float) : int
+queryStockValueByID(stockID : int) : QueryResult
+sellStock(accountID : int, stockID : int, amount : int) : void
+updateAccount(accountID : int, credit : float) : void
+getHoldingStatement(accountID : int, start_stockID : int) : Collection
...
Seeder
<<TestContext>>
ClientDemo
+loadTestAll() : void
Trxn Data
+tranType : int
+accountID : int
+stockID : int
+holdingID : int
+amount : int
+start_stockID : int
+updateCredit : float
+address : String
+name : String
+credit : float
<<DataPool>>
Tran Deck
+@TRAN_BUY : int = 1
+@TRAN_GETHOLDING : int = 5
+@TRAN_QUERY_CODE : int = 4
+@TRAN_SELL : int = 2
+@TRAN_QUERY_ID : int = 3
+@TRAN_CREATE : int = 6
+@TRAN_UPDATE : int = 7
+@SIZE : int = 43
+shuffle() : void
<<Data Partition>>
LoadTestingTrxn Data
+getLoadTestingTrxnData() : Trxn Data

Figure 2. Stock-Online benchmark load test model

4. CONCLUSION

We have validated our tool through a case study. We took an existing benchmark application which took considerable time to be built and performance tested for different middleware platforms, and undertook the same task using our tool. One student, the main developer of the AndroMDA extension but with no experience with J2EE, took one week to model the system and conduct the load tests on WebLogic. It took her another half day to deploy the system on JBoss and conduct the same load testing. Hence we believe the productivity savings afforded by this approach are considerable. The effort involved in the basic plumbing code and load test suite has been absorbed by cartridge developers and code generation. Making changes to the design to generate a new version of the benchmark is also simpler. We are currently migrating the tool onto other platforms with other flavors of Model Driven Development (MDD) such as .Net and DSL-based MDD.

5. REFERENCES

[1] M. J. Rutherford and A. L. Wolf, "A case for test-code generation in model-driven systems," in Proceedings of the The second international conference on Generative programming and component engineering, Erfurt, Germany, 2003.

[2] 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.

[3] I. Gorton and A. Liu, "Evaluating the performance of EJB components," *Internet Computing, IEEE*, vol. 7 (3), pp. 18-23, 2003.

[4] "AndroMDA, v3.0M3", http://andromda.org/.

[5] "Model Driven Architecture", http://www.omg.org/mda/.

[6] "UML 2.0 Testing Profile Specification", http://www.omg.org/cgi-bin/doc?ptc/2004-04-02.