Generating Performance Test Scripts and Scenarios Based on Abstract Intermediate Models

Conference Paper · January 2012

DOI: 10.13140/RG.2.1.1033.7445

Leandro T. Costa
Pontifícia Universidade Católica do Rio Grande do Sul
7 PUBLICATIONS **62** CITATIONS
SEE PROFILE

Ricardo M. Czekster
Aston University
86 PUBLICATIONS **249** CITATIONS
SEE PROFILE

Flávio M De Oliveira
43 PUBLICATIONS **228** CITATIONS
SEE PROFILE

Elder Rodrigues
Universidade Federal do Pampa
60 PUBLICATIONS **157** CITATIONS
SEE PROFILE

Generating Performance Test Scripts and Scenarios Based on Abstract Intermediate Models

Leandro T. Costa, Ricardo M. Czekster, Flávio M. de Oliveira,
Elder M. Rodrigues, Maicon B. da Silveira, Avelino F. Zorzo
Faculty of Informatics (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil
Email: leandro.teodoro@acad.pucrs.br, ricardo.czekster@pucrs.br, flavio.oliveira@pucrs.br,
elder.rodrigues@pucrs.br, bernardino@acm.org, avelino.zorzo@pucrs.br

***Abstract*—Performance testing involves knowledgement not only about the application to be tested, its usage, and the execution infrastructure; it also requires understanding of the performance test automation tools employed – scripting, monitoring and configuration details. Performance script generation is highly technology-dependent, with great variations from one test engine (the workload generator) to another. The scenarios, however, should not depend on that. In this paper, we present a technology-independent format for test scenarios and test cases, henceforth denominated *abstract intermediate models*, and the process for deriving specific test scripts from them. Hence, we can easily reuse test scenarios with different workload generators, allow the performance engineers to edit the scenarios before the script generation, and abstract details related to configuration of workload generators and monitors.** [1] [2]

I. Introduction

Performance testing is a highly specialized task. It involves knowledgement about the application to be tested, its usage profile, and the infrastructure where it will operate. Furthermore, because it involves intensive test automation, performance testing requires understanding of the performance test automation tools that will be used. Hence, there is a bottleneck in productivity of performance engineering teams, due to the increasing complexity of the performance testing tools and the workload to generate the test case/scripts.

An approach that could be used to automatically generate performance test cases/scripts is Model-based Testing (MBT) [8]. MBT can be used not only to automatically derive performance test scenarios from the System Under Test (SUT) models, but also to automate the test execution. The SUT could be modelled under a wide range of modeling languages, such as state machine diagrams [9], Specification and Description Language (SDL) [12] and Unified Modeling Language (UML) [3]. Probably, the most widely used modeling language in the industry is UML. The UML provides a notation for modeling some important characteristics of applications, allowing the development of automatic tools for model verification, analysis and code generation. Performance is one of these characteristics; it is the object of the UML Performance Profile [10], which defines a standard way to represent performance information in a UML model.

In previous works [5] [15], we described our technique for deriving the test scenarios and the corresponding scripts from UML diagrams. During experiments, we learned that script generation is highly technology-dependent, with great variations from one test engine (called workload generator) to another. The scenarios, however, should not depend on that, because they should be reused to generate scripts for several workload generators. Thus, it would be wise to separate technology details from the process of generating test scenarios.

Information Technology (IT) companies, specially global companies, frequently use different workload generators, due to costs, marketing and/or different target execution platforms. The reuse of the scenarios, combined with automatic script generation, reduces overall testing effort. In addition to contribute to a technology migration, the test scenarios could have test information in a clear format and common to many technologies. Therefore, this information could be easily used to generate scripts to a wide range of performance tools, e.g. LoadRunner [11] and Visual Studio [13].

In this paper, we describe our approach to generate test scenarios and test cases, which could be used to derive scripts to a wide range of workload generator. In order to accomplish that, we defined a technology-independent format for test scenarios, which we call *abstract intermediate models*, and the process for deriving specific test scripts from them. Then, we can easily reuse test scenarios with different workload generators, also allowing the performance engineers to edit the scenarios before the script generation and abstract details related to configuration of workload generators and monitors.

The structure of this paper is organized as follows. Section II discusses related background. Section III explains the conversion from UML diagrams to test scenarios, discussing major trade-offs and advantages. Section IV provides an example demonstrating the abstract test scenarios that were derived automatically from the UML diagrams.

II. Background

Performance is a fundamental quality of software systems, affecting all underlying layers of systems. Therefore, in order

[1]Study developed by the Research Group of the PDTI 001/2012, financed by Dell Computers of Brazil Ltd. with resources of Law 8.248/91.

[2]The order of the authors is merely alphabetical.

to improve the software quality of applications, Software Performance Engineering (SPE) is used to describe and provide means to improve the performance through two distinct approaches: early cycles studies based on predictive models and late cycles inspections based on measurements [18]. One SPE activity is performance testing, responsible to perform tests for part or for the entire system, under normal load and/or stress load. Nowadays, there are some techniques, e.g., Record/Playback [14] and Model-based Testing, that are used by some tools to automate the performance test case generation and results analysis.

MBT [4] [8] [17] is a technique to test systems in which models are designed to create testing models suitable mapping a given reality a model is a useful way for into analyzing quality properties and performance attributes as it eases understanding and promotes divide-and-conquer rationale [1]. Thus, a SUT model can be constructed anticipating problems that may or may not arise in early design. However, the main usage of MBT has been directed to the test of functional aspects (defects) of software and only lately researchers are applying some of its techniques towards non-functional testing.

Therefore, lately, an increasing numbers of works [2] [5] [6] [15] discuss how to apply MBT and UML models to automatic generate performance test cases and scripts. Most of them apply MBT in conjunction with MARTE Profile [10], an extension of UML for modeling performance requirements. Its purpose is to provide modeling patterns with clear semantics, allowing automated performance model analysis. Based on the UML models, one can extract information for the tests and then analyse the performance counters (e.g. throughput, transactions per second and response time). Such information is distributed throughout several UML diagrams, e.g., use cases (UC), activity (AD) and sequence diagrams. An approach that depicts how this diagrams are used to represent the information of performance testing is presented by [6], in which a model designed in MARTE and composed by UML diagrams was tagged with specific properties such as probabilities on the incidence of use cases, response times for each activity, resource availability, etc.

III. UML Diagrams and Test Scenarios

The common first step adopted by the industry when testing applications for non-functional properties is to choose a given load engine among the various available tools. The decision on which tool to best test an application involves several factors such as familiarity (has been used before), pricing or target software platform.

However, every workload manager has its own peculiarities, configurations and tweaks as well as common operations to be performed when mimicking user behavior within the SUT. Thus, the test scenarios and scripts generated for some tool can not be reused to another.

Figure 1 shows our approach, in which the main idea is to convert high-level UML diagrams to an abstract intermediate model suitable for the derivation of abstract test scenarios and scripts to different workload manager. The main difference between our approach and classic MBT abstract test suites is the fact that we are interested in creating intermediate models to use for performance testing rather than functional testing. Another contribution is that we apply MBT to construct a performance abstract test scenario. Based on that, can be generated test scenarios and scripts for a wide range of workload manager.

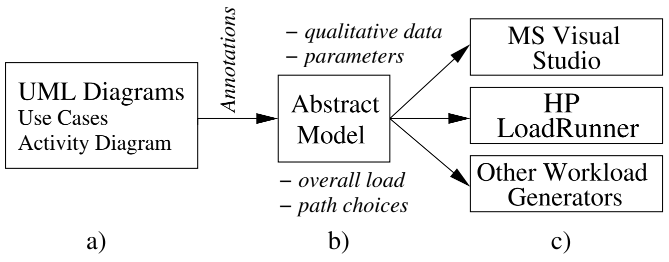

Fig. 1. Model transformation engine

A. Abstract Intermediate Models

Our focus in this paper is generate a technology-independent testing description from a MBT approach. Thus, we reuse the applications models to generate an abstract intermediate model that is a description of the test scenarios. It contains information needed to instantiate the performance scripts, such as loads, ramp-up time, total execution time etc., which can be derived from the model, while abstracting script details such as machine locations, API calls etc. An important issue when using MBT to generate performance test scenarios and scripts is define which UML diagram and which UML profile will be used to annotate the needed information.

B. Transformation Methodology

This section describes how to convert UML models (mostly UC and AD) into general purpose abstract intermediate models. Before describing the overall methodology, we discuss the UML annotation provided by other parties. Our approach relies on the assumption that the UML models are carefully annotated with high-quality information. We are trusting that all stakeholders are interested in the process of discovering the application's problems and bottlenecks and also that all involved personnel is committed to create comprehensive test scenarios, allowing further reliable analysis.

The process of generating test scenarios from annotated UML models must encompass a set of operations that must be performed prior to the test execution. Our methodology involves basically three steps:

1) UML Annotations Verification (Figure 1-a): We focus our attention to two broadly used diagrams present in UML: Use Cases (UC) and Activity Diagrams (AD). This step entails the evaluation if the models were annotated with information that can be used to the generation of the abstract model in the next step. Thus, to annotate these information on the UML models we defined some UML tags based on the UML profile to Schedulability Performance and Time (SPT) [18]: a) <<**TDtime**>>: limits the performance test duration(s) for the scenario(s); b) <<**TDpopulation**>>: maps the number of virtual users that populates the system; c) <<**TDhost**>>: describes the name of the host to connect to; d) <<**TDaction**>>: specifies an action that must be taken, e.g., the execution of a script; e) <<**TDparameters**>>: represents two information: name and value, that must be filled out to proceed, e.g., a login screen, or a search form; f) <<**TDprob**>>: indicates the probability to decide the next activity to be executed. It is used in the decision element model within the ADs or annotated in the association element model between actor and use case within the use cases diagram; g) <<**TDthinkTime**>>: defines the amount of time units each virtual user waits before taking another action. This tag is used by a large set of workload generators to mimic user behavior.

After this step, the UML models should satisfy the following conditions: a) every UC have at least one or several ADs counterparts; b) the AD is well-formed, i.e., contains an initial and an end state; c) test plan with information regarding the test itself, e.g., the type of test, the number of virtual users; d) every activity is annotated with context information, i.e., user form and query string parameters and user think time information.

2) Intermediate Model Generation (Figure 1-b): Our abstract intermediate model is designed to be extensible and modular, using hierarchical textual structures to represent activities readily available in UML diagrams or other similar representations of business processes (e.g. Business Process Model Notation (BPMN) [7]). In fact, it is straightforward to take any high-level business process representation and directly transform it to a test scenario, however, the test must be instantiated having a specific workload generator in mind.

This step is split in two abstract models: abstract test scenarios and abstract test cases. Each abstract models is a hierarchical structure that is used to map the AD to a textual abstract representation retaining the order of events and the parallel activities that must take place. This structure uses a sequential numbering convention with appended activities, e.g., 1, 1.1, 2.1 and so forth. Each activity is mapped to a number, here defined as an *activity number*. The numbered mapping is a visual aid to inspect sequential and parallel relations within the abstract intermediate model.

Note that for the abstract model to function according to our specification, it should contain only the fundamental information to instantiate different test scenarios. The abstract format suggested here is extensible and could be enhanced to contain more information regarding performance tests for a more complex test suite.

3) Workload Generator Instantiation (Figure 1-c): This step is tool-oriented because it generates specific test scenarios that must be created for each abstract intermediate model. We explain how this is performed in the following sections. Our approach shifts the concern on the performance test execution itself to the description of an abstract model that captures the needed test information looking up only to high-level (UML models). Next, we present how to apply our approach to generate scripts to two workload generators.

IV. Example of Use: Generating Scripts Based on Abstract Test Scenarios

This section describes an application scenario which our approach is applied to generate abstract models. For this purpose, we used the TPC-W benchmark [16] as an application example and develop a tool to generate automatically the abstract scenarios and then generate scripts for MS Visual Studio and LoadRunner. The TPC-W is a transactional web service benchmark that implements a benchmark and an e-commerce application that is used by the benchmark (or by any other workload generator).

To support the generation of scripts to test the TPC-W application, we developed an application to parse the information annoted in the UML models and generate the abstract models (following the guidelines described in Section III). This application is derived from the Software Product Line called PLeTs [5] [15].

A. TPC-W UML Diagrams

The first step to apply our approach (Section III) to test the TPC-W application is to annotate the TPC-W models. For this task, we have created several UML based tags to represent the performance information needed to generate the abstract intermediate models. As a starting point we have annotated three different use cases (shown in Figure 2): a) `Browser`: the users performs browsing interactions; b) `Registration`: the user fulfill a registration form; and c) `Shop`: the users performs purchase interactions.

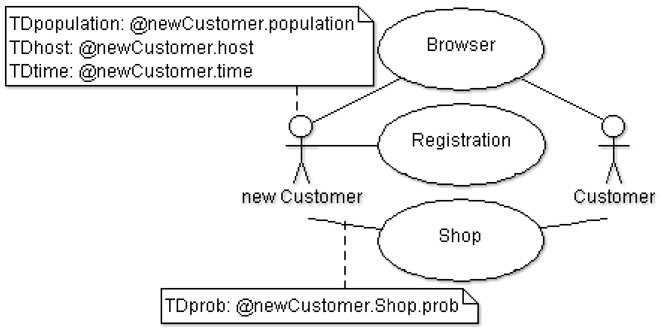
TDpopulation: @newCustomer.population
TDhost: @newCustomer.host
TDtime: @newCustomer.time
Browser
Registration
new Customer
Customer
Shop
TDprob: @newCustomer.Shop.prob

Fig. 2. TPC-W Use Case Diagram

Each actor present in the UC diagram contains information for a specific test scenario. Thus, we define two different test scenarios interactions for actors `Customer` and `New Customer`. The test scenario for `New Customer` is a set of interactions that will be performed over the set of use case elements (`Browser, Registration` and `Shop`). It is important to notice that each UC is related to a specific AD, e.g., the AD present in the Figure 3 is related to the `shop` use case (Figure 2). Basically, each AD represents the sequence of activities performed by an actor over the application. As depicted in Figure 3, the first activity is `Home Page`, which is the TPC-W home page. After that, the user could

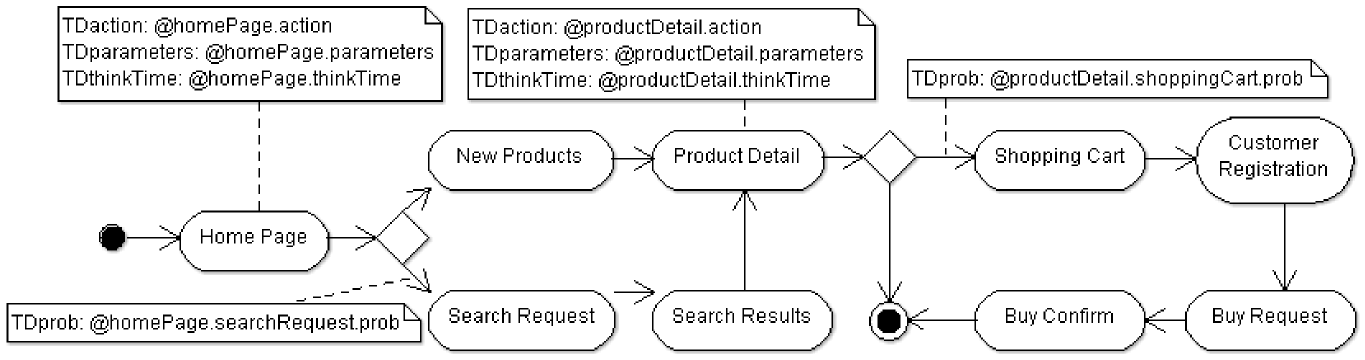
TDaction: @homePage.action
TDparameters: @homePage.parameters
TDthinkTime: @homePage.thinkTime
TDaction: @productDetail.action
TDparameters: @productDetail.parameters
TDthinkTime: @productDetail.thinkTime
TDprob: @productDetail.shoppingCart.prob
New Products
Product Detail
Shopping Cart
Customer Registration
Home Page
TDprob: @homePage.searchRequest.prob
Search Request
Search Results
Buy Confirm
Buy Request

Fig. 3. TPC-W Activity Diagram

perform the following actions: selects a specific book category (`New Products`) or performs a search for a particular book (`Search Request` and `Search Results`). The activity `Search Request` shows as a result a list of books to the user. Hence, when selecting a book from this list (`Search Results`), several information of the selected book are displayed to the user (`Product Detail`). After that, the user must perform one of the following activities: finish his access to the application or continue on the website and make a purchase (`Shopping Cart`). If the user decided for the latter option, the next step is related to the registration of customer (`Customer Registration`) in the application. Then, the user fills some information of purchase such as financial information (e.g. credit card number) and delivery date (`Buy Request`). The last step checks if all information is correct, then the purchase is confirmed (`Buy Confirm`) and finishes the access to the application. This diagram has also two decision elements in its flow, that represent the probability of executing different paths in the system, e.g., the tag `@homePage.searchRequest.prob` between activities `Home Page` and `Search Request` in the Figure 3.

As described in Section III, the UML diagrams can be initially annotated with seven tags in our approach. The UC shown in Figure 2, has four tags: `TDpopulation`, `TDhost`, `TDtime` and `TDprob`. Each one has its respective parameter, `@customer.population`, `@customer.host`, `@customer.time` and `@newCustomer.BrowsingMix.prob`. In relation to our AD we have also included four different tags and their respective parameter (see Figure 3).

B. Abstract Test Scenarios Generation

Once all the UML diagrams (see Figures 2 and 3) were annotated with performance informations, we apply our approach to generate abstract test scenarios for the TPC-W application. The creation of abstract test scenarios allows a later definition of a workload manager and its test script templates.

Basically, an abstract test scenario defines the amount of users that will interact with the SUT and also specifies the users behavior. The abstract test case contains the information regarding the necessary tasks to be performed by the user.

Figure 4 shows an example of an abstract test scenario generated for the actor `Customer`. The amount of abstract test scenarios generated based on a UC diagram is directly related to the amount of actors modeled in the UC model, e.g., for the TPC-W example there are two abstract test scenarios.

```
Abstract Test Scenario: Customer
## Test Setting
Virtual Users : <<TDpopulation: @customer.population>>
Host of SUT : <<TDhost: @customer.host>>
Test Duration : <<TDtime: @customer.time>>
## Test Cases Distribution:
<<TDprob: @customer.Shop.prob>>
1. Shop
1.1. Shop 1
1.2. Shop 2
1.3. Shop 3
1.4. Shop 4
<<TDprob: @customer.Browser.prob>>
2. Browser...
<<TDprob: @customer.Registration.prob>>
3. Registration
3.1. Registration 1...
3.4. Registration 4
```

Fig. 4. Abstract test scenario of the actor `Customer`

As presented in Section III, the abstract test scenario has the information related to the test context and the definition of the abstract test cases that must be instantiated, including the distribution of the number of virtual users for each abstract test case. Thus, our annotation approach is divided in two groups: 1) *Test Setting* – describes the general characteristics that are applied to the test context as a whole (extracted from the UC); 2) *Test Cases Distribution* – represents information specific to the abstract test cases generated from each AD. In order to accomplish that, each test case represent a user path in the SUT. It is important to notice that the header of each abstract test case contains probability information (see Figure 4) .

As show in Figure 5, the abstract test cases are built in a hierarchical approach, in which activities are listed and organized according to the dependency between the AD activities (represented by *activity number*). Figure 5 shows the abstract test case based on one test sequence derived from the AD (Figure 3). Furthermore, in the description of abstract test cases there is a variation of parameters added to each activity, which are composed by the name and the value of each tag, showing the flexibility of the configuration models. A parameter is the concatenation of two pieces of information: activity name and tag name, preceded by the delimiter `@`. Although, the tag `TDprob` has three pieces of information. The first of these information is the tag name preceded by the name of two UML elements. An example of that is presented in Figure 3 where the tag `TDprob @homePage.searchRequest.prob` is tagged in the UML association element between the UML decision node and the target activity (`Search Request`). The same notation is applied in the UC diagrams, where the tag is applied between the UC `Shop` and the actor `new Customer`.

```
#Abstract Test Case: Shop 3
1. Home Page
   <<TDmethod : @HomePage.method>>
   <<TDaction : @HomePage.action>>
   <<TDparameters : @HomePage.parameters>>
   <<TDthinkTime : @HomePage.thinkTime>>
2. New Products...
3. Product Detail...
4. Shopping Cart...
5. Customer Registration...
6. Buy Request...
7. Buy Confirm
   <<TDmethod : @BuyConfirm.method>>
   <<TDaction : @BuyConfirm.action>>
   <<TDparameters : @BuyConfirm.parameters>>
   <<TDthinkTime : @BuyConfirm.thinkTime>>
```

Fig. 5. Example of abstract test case generated from the `Shop` use case

It is important to notice that each tag parameter refers to a data file (Figure 6), that is automatically generated. Thus, when abstract test scenario and scripts are instantiated to a concrete test scenario and scripts for a specific workload generator, the tag parameter is replaced by a value extract from a file.

```
@BuyConfirm.method:"POST"
@BuyConfirm.action:"http://localhost/tpcw/buy_confirm"
@BuyConfirm.parameters:[$ADDRESS.CITY, $ADDRESS.STATE]
@BuyConfirm.thinkTime:5
```

Fig. 6. Example of data file containing some parameters

C. Test Scenarios and Scripts Instantiation

Based on the abstract test scenarios and test cases presented in Section IV-B, the next step is to generate concrete instances (scripts and scenarios). This is a technology dependent step, because the concrete scenarios and test cases are strongly related to a specific workload generator that will directly execute the test cases.

This is an important step on the automation of the performance testing, because it allows the flexibility of choice a workload generator or technology only in the execution stage of the test cases. However, it is necessary an advanced tool knowledgement to create scripts and scenarios. Therefore, to demonstrate how our approach could be valuable to a performance testing team, we presents how to generate test scenarios and scripts for two workload manager: Visual Studio (VS) and LoadRunner (LR). Basically, the VS and the LR structure its test scenarios and scripts in two files. One of them is a scenario file that is used to store the test configuration, workload profile distribution among test scripts and the performance counters that will be monitored by the tool. The other file is a script configuration file that is used to store the information about users' interaction with the application, including HTTP requests, as well as its parameters and transactions defined between requests. Figure 7 shows the VS scenarios file that was generated to test TPC-W. In this case, the `MaxUser` property corresponds to the parameter `@customer.population`. Another tag that has changed was the `RunConfiguration` with attribute `RunDuration` that is related to the tag `@customer.time`. The process to instrument a test scenario is based on a template. Hence, the further information within the scenario, that are not from the abstract test scenario are those standard information present in any test scenario generated by the workload generator.

```
<LoadTest ...>
  <Scenarios>
    <Scenario Name="Customer" ...>
      <ThinkProfile Value="0" Pattern="On"/>
      <LoadProfile Pattern="Step" InitialUsers="0"
           MaxUsers="50" StepUsers="10" StepDuration="0"
           StepRampTime="60"/>
      <BrowserMix>...</BrowserMix><TestMix>...</TestMix>
      <NetworkMix>...</NetworkMix>
    </Scenario>
  </Scenarios>
  <CounterSets>...</CounterSets>
  <RunConfigurations>
    <RunConfiguration RunDuration="7200" WarmupTime="300"
        TestIterations="100" ...>...</RunConfiguration>
  </RunConfigurations>
</LoadTest>
```

Fig. 7. XML of test scenario generated for the Visual Studio (*.LoadTest)

Figure 8 presents a snippet of VS test scripts that was generated to test TPC-W. In turn, Figure 9 shows a snippet for LR. These test scripts were instantiated based on abstract test case presented in Figure 5. Basically, the test scripts are a set of several HTTP requests. Among the features correlated in the set of test artifacts, we highlight the following example: 1) `Tag Request` - in the VS the attribute `Url` and the attribute `web_submit_data` in the LoadRunner are related to the parameter `@BuyConfirm.action`; the VS attribute `ThinkTime` and the parameter `lr_think_time` LoadRunner are correlated to the parameter `@BuyConfirm.thinkTime`; 2) `Tag QueryStringParameter` - the VS and LoadRunner attributes `Name` and `Value` are related to the parameter `@BuyConfirm.parameters`.

V. Final Considerations

The present work proposed a common format to define abstract intermediate models suitable for the instantiation of

```
<WebTest Name="Shop 3"...>
 <Items>
  <TransactionTimer>...</TransactionTimer>
  <TransactionTimer Name="Buy Confirm">
   <Items>
    <Request Url="http://localhost:8080/tpcw/
         TPCW_buy_confirm_servlet" ThinkTime="5"...>
     <QueryStringParameters>
       <QueryStringParameter Value="{{$ADDRESS.CITY}}"
            Name="CITY".../>
       <QueryStringParameter Value="{{$ADDRESS.STATE}}"
            Name="STATE".../>...
     </QueryStringParameters>
    </Request>
   </Items>
  </TransactionTimer>
 </Items>
 <ValidationRules>...</ValidationRules>
</WebTest>
```

Fig. 8. Test script generated for the Visual Studio

```
Action()
{
  ...
  lr_think_time(5);
  web_submit_data("buy_confirm.jsp",
  "Action=http://localhost:8080/tpcw/TPCW_buy_confirm_servlet
       ",
  "Method=POST",
  "RecContentType=text/html",
  "Referer=",
  "Mode=HTML",
  ITEMDATA,
    "Name=CITY", "Value={{$ADDRESS.CITY}}", ENDITEM,
    "Name=STATE", "Value={{$ADDRESS.STATE}}", ENDITEM,
  LAST);
  ...
}
```

Fig. 9. Test script generated for the LoadRunner

test scenarios for different workload managers. Throughout the paper we have discussed some important aspects on how to use annotated UML models to derive an intermediate textual format having the most important primitives that are needed to construct comprehensive test scenarios. We also show how to transform the abstract test scenarios in test script for two workload manager.

Our technique provides an indication that generating abstract models is a powerful means to derive effective technology-independent test scenarios. It is important to highlight that the creation of an abstract model for later definition of a test script and scenario using a chosen workload manager needs only to annotate a few selected data in the UML models. Translating UML models to this representation is also more comprehensible for end-users when they are tracking bugs or trying to understand the flow of operations for a functionality.

We envision several future works to consider following the present proposition. One could, for example, seamlessly translate a different UML model (e.g. Sequence Diagram) using our abstract model to generate scripts to some tool that is based on a different testing technique, e.g, structural testing. Another concern that has come to our attention is directed to the description of an abstract model to relate more architectural information in terms of the underline infrastructure of the SUT, for instance, the use of virtualized environments or cloud computing.

Acknowledgments. We thank CNPq/Brazil, CAPES/Brazil, INCT-SEC, and DELL for the support in the development of this work.

References

[1] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 30:295–310, May 2004.

[2] C. Barna, M. Litoiu, and H. Ghanbari. Model-based performance testing (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, pages 872–875, New York, NY, USA, 2011. ACM.

[3] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide (2nd Edition)*. Addison-Wesley Professional, 2005.

[4] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In *Proceedings of the 21st International Conference on Software engineering*, pages 285–294, New York, NY, USA, 1999. ACM.

[5] E. de M. Rodrigues, L. D. Viccari, and A. F. Zorzo. Plets-test automation using software product lines and model based testing. In *22th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 483–488, 2010.

[6] S. Demathieu, F. Thomas, C. Andre, S. Gerard, and F. Terrier. First experiments using the uml profile for marte. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 50–57, may 2008.

[7] R. M. Dijkman, M. Dumas, and C. Ouyang. Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12):1281–1294, 2008.

[8] I. K. El-Far and J. A. Whittaker. *Model-based Software Testing*. Wiley, New York, 2001.

[9] R. Ferreira, J. Faria, and A. Paiva. Test Coverage Analysis of UML State Machines. In *Proceedings of the 3rd International Conference on Software Testing, Verification, and Validation Workshops*, pages 284 –289, april 2010.

[10] O. M. Group. UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE). *MARTE specification version 1.0. OMG, 2009. OMG document number formal/2009-11-02.*, 2009.

[11] Hewlett Packard - HP. *Software* HP LoadRunner, Sep. 2010. URL: https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-126-17\^8_4000_100.

[12] A. Kerbrat, T. Jéron, and R. Groz. Automated test generation from sdl specifications. In *Proceedings of the 6th SDL Forum*, pages 135–152, 1999.

[13] J. Levinson. *Software Testing With Visual Studio 2010*. Pearson Education, 2011.

[14] G. Meszaros. Agile regression testing using record & playback. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 353–360, New York, NY, USA, 2003. ACM.

[15] M. B. Silveira, E. M. Rodrigues, A. F. Zorzo, L. T. Costa, H. V. Vieira, and F. M. Oliveira. Generation of Scripts for Performance Testing Based on UML Models. In *23rd International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 1–6, 2011.

[16] TPC-W Org. Benchmark TPC-W, Feb. 2012. URL: http://http://www.tpc.org/tpcw.

[17] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann, San Francisco, 2006.

[18] C. Woodside and D. Petriu. Capabilities of the UML Profile for Schedulability Performance and Time (SPT). In *Workshop SIVOES-SPT RTAS'2004*, 2004.