Extending UML Testing Profile Towards Non-functional Test Modeling

Federico Toledo Rodríguez[1,3], Francesca Lonetti[2], Antonia Bertolino[2], Macario Polo Usaola[3] and Beatriz Pérez Lamancha[3]

[1]*Abstracta, Montevideo, Uruguay*

[2]*CNR-ISTI, Pisa, Italy*

[3]*Alarcos Research Group, UCLM, Ciudad Real, Spain*

ftoledo@abstracta.com.uy, {francesca.lonetti, antonia.bertolino}@isti.cnr.it, {macario.polo, beatrizp.lamancha}@uclm.es

Keywords: Model-based Testing, Non-functional Test Cases, UML-TP.

Abstract: The research community has broadly recognized the importance of the validation of non-functional properties including performance and dependability requirements. However, the results of a systematic survey we carried out evidenced the lack of a standard notation for designing non-functional test cases. For some time, the greatest attention of Model-Based Testing (MBT) research has focused on functional aspects. The only exception is represented by the UML Testing Profile (UML-TP) that is a lightweight extension of UML to support the design of testing artifacts, but it only provides limited support for non-functional testing. In this paper we provide a first attempt to extend UML-TP for improving the design of non-functional tests. The proposed extension deals with some important concepts of non-functional testing such as the workload and the global verdicts. As a proof of concept we show how the extended UML-TP can be used for modeling non-functional test cases of an application example.

1 INTRODUCTION

In modern software systems the validation of non-functional properties, such as performance and dependability, becomes more and more important, so that a variety of automated approaches and tools for verification and assurance of non-functional requirements are being proposed. The main aim of these approaches is to evaluate the system performance in terms of responsiveness and stability under a particular workload or to assess other quality attributes of the system, such as scalability, dependability and resources usage.

Concerning the modeling of non-functional properties, numerous profiles have been proposed to allow easy and accurate design of performance and dependability aspects of modern software systems. Among them, MARTE (OMG, 2011) extends UML (OMG, 1997) by providing a rich framework of concepts and constructs to model non-functional properties of real-time and embedded systems and defines annotations to augment models with information required to perform quantitative predictions and analysis of time-related aspects, such as schedulability and performance.

An important role in the software validation process is played by Model-Based Testing (MBT) (Utting and Legeard, 2007), which contributes to test automation by pushing the application of model-based design techniques to software testing. It involves the development of models that describe test cases, test data and the test execution environment, and the application of automated facilities for generating executable test cases from these models. A key element involved in MBT is the modeling language used for defining a test model from the informal system requirements or the design models. For testing purposes, it is important to have a test model that is easy to verify, modify and manipulate without losing all the information needed to generate test cases.

The authors of (Dias Neto et al., 2007) present a systematic literature review on MBT. A result of this review is that by adopting models developed from the analysis of the abstract behavior of the System Under Test (SUT), MBT has been traditionally used for generating functional test cases, but has missed to address non-functional requirements.

The first contribution we present in this paper is a systematic survey on MBT approaches for non-functional requirements. However, as a result of this review, we noticed a lack of a standard and common language for designing non-functional test cases. The

only proposal in this direction is the UML Testing Profile (UML-TP) (Baker et al., 2007), a standard notation to support the design, visualization, analysis and documentation of the artifacts involved in testing. However, UML-TP provides only limited support for non-functional testing and we noticed that it does not allow specifying some important concepts of performance testing such as the workload and the definition of a global verdict for concurrent executions.

This paper aims to overcome this limitation by providing insights about the design of non-functional aspects with UML-TP. Specifically, the main addressed issues deal with the representation of the workload and the verdict definition involved into non-functional validations. For the first one, we introduce a new stereotype called "Workload", which can specify among others, the amount of concurrent executions, the intensity of the execution, the test duration and the ramp-up. Concerning the non-functional validation, we propose to extend UML-TP with the ability to express global verdicts, considering also the average (or percentage) of the response times of all the executions of a given test case in a workload simulation. To illustrate the usefulness of our proposal, we provide an application example in which we define some non-functional requirements and show how to apply the extended UML-TP for designing test cases for such requirements.

The remainder of this paper in organized as follows. Section 2 contains the results of a systematic survey addressing model-based testing approaches for non-functional requirements. Section 3 gives an overview of the current version of UML-TP outlining its limitations for designing performance and dependability tests. Section 4 presents some insights for extending UML-TP whereas Section 5 shows their application for defining a load test modeling example. Finally, Section 6 concludes the paper, also hinting at future work.

2 SYSTEMATIC SURVEY

To know the state of the art about model-driven approaches for non-functional validation we performed a systematic literature review. This survey has been conducted following the guidelines for systematic review in software engineering research proposed by Kitchenham (Kitchenham, 2004). These guidelines cover three aspects of a systematic review: planning the review, conducting the review and reporting results. Below we briefly present first the research method and then the obtained results.

2.1 Research Method

In the planning phase of our systematic survey we identified the following research question (RQ):

RQ: what approaches have been proposed for non-functional test modeling and automatic generation of non-functional test cases with model-driven approaches?

According to this research question we defined the following search string:

```
((test or verification or testing or
            validation)
                AND
 (performance or "non-functional" or
   nonfunctional or dependability)
                AND
(model or metamodel or "meta model" or
  "model-driven" or "model-based"))
```

We searched by title in the IEEE Xplore, ACM Digital Library, SCOPUS, WEB OF KNOWLEDGE databases, and selected "English papers" from 1990 to 2013 (30 July):

From this automatic search we obtained 411 papers (157 in IEEE Xplore, 67 in ACM Digital Library, 103 in SCOPUS, 84 in WEB OF KNOWLEDGE), reduced to 25 after reading the title, keywords and abstract, and finally reduced to 24 papers after reading the full text.

2.2 Results

For the sake of space we do not present here the overall results of the systematic survey but we focus on the most relevant works about model-driven validation approaches of non-functional properties. We classified them according to four main research directions: benchmark generation; performance tests generation; models to predict performance and Software Performance Engineering (SPE); search-based testing for non-functional properties.

We refer to (Rodríguez et al., 2013) for a more complete report about the systematic survey results.

Benchmark Generation. There are a large number of code generation techniques that can be used in benchmark suite generation. The aim of the work in (Zhu et al., 2007) is to automate the generation of a complete benchmark suite from a UML-based design description, along with a load testing suite modeled in the UML-TP. The authors have tailored UML-TP to represent a workload including some tagged values such as the number of process the load generator should start, the number of threads that each process spawns, the maximum length of time in millisec-

onds that each process should run for, the time interval between starting up or stopping new processes, etc. Thus, the main result of the approach is the automated generation of the application under test including a complete test harness with the load to execute and some facilities to automatically collect monitoring information. However, differently from our work, the effort of the authors is more focused on automated benchmark generation than on test modeling. They are merely concerned on representing load tests using the tailored profile in order to produce a configuration file containing all the tagged values and derive a default implementation of the model including both test logic and test data.

In (Cai et al., 2004) a performance test-bed generator for industrial usage is proposed. The test bed is modeled indicating interactions between client and server that have to be simulated. The model is used to generate the code of a performance test bed able to run the specified performance tests. Differently from our proposal, this approach does not focus on the load test modeling, but it aims at solving some challenges of performance test bed generation including the extension of the open-source ArgoUML tool to provide UML-like architecture modeling and XMI-derived model representation capabilities.

Performance Tests Generation. A recent research direction in performance testing is the model-based generation of test-beds for assessing that the application meets its performance requirements. In particular, the approach in (de Oliveira et al., 2007) takes as input a performance specification using the UML 2.0 SPT Profile, and derives a modified Stochastic Petri Net from which a realistic test scenario in JMeter (Apache, 2001) is generated. It also has a result interpreter to give a verdict. The main difference with our proposal is that this approach tries to generate the test code directly from the requirements specification without having a test model that is the objective of our work. Another solution in this context is presented in (da Silveira et al., 2011) in which the authors propose to include five stereotypes in the system UML models (use cases and activity) to express performance information that will be used for generating test scripts for a commercial tool called LoadRunner (Mercury, 2001). Specifically, these stereotypes include: *PApopulation* representing the number of users and the host where the application is executed, *PAprob* representing the probability of execution for each existing activity, *PAtime* representing the expected time to perform a given use case, *PAthinktime* that is the time between two different user actions and *PAparameters* representing the input data to be provided to the application when running the test scripts. The main limitation of this approach is that it addresses the issues of a specific performance testing tool that is LoadRunner.

An attempt to model the workload for performance test generation is in (Krishnamurthy et al., 2006). It provides a tool for generation of a synthetic workload characterized by sessions of interdependent requests. From requests logs of a system under test, the approach automatically creates a synthetic workload that has specified characteristics and maintains the correct inter-request dependencies. The main difference of this approach with respect to our proposal is that it pays much attention to the scripts generation and the definition of how to take data (from an external file of from the previous response) than to the workload specification.

Finally, another performance tool is proposed in (Abbors et al., 2012; Abbors et al., 2013). This tool aims to evaluate the performance of a system and monitor different key performance indicators (KPI) such as the response time, the mean time between failures, the throughput, etc. The tool accepts as input a set of models expressed as probabilistic timed automata, the target number of virtual users, the duration of the test session and will provide a test report describing the measured KPIs. The main contribution of the paper is that the load applied to the system is generated in real time from the models.

Moreover, four metamodels related with performance testing are presented in (Pozin and Galakhov, 2011). They are: metamodel of requirements, metamodel of the system, metamodel of the load, metamodel of measurements. The use of these metamodels in planning a new load testing experiment make it possible to automate the configuration of automated testing tools for the parameters of a specific load experiment.

Models to Predict Performance and Software Performance Engineering (SPE). An orthogonal research direction to our work is represented by model-based software performance prediction. An extensive survey on methodological approaches for integrating performance prediction in the early phases of the software life cycle is presented in (Balsamo et al., 2004). This survey gives some indications concerning the software system specification and the performance modeling. For software specification, most of the analyzed approaches use standard practice software artifacts like UML diagrams whereas Queueing Network Model and its extensions are candidate as performance models since they represent an abstract and black box notation allowing easier model comprehension. However, a performance model inter-

change format (PMIF) (Smith et al., 2010) has been proposed as a common representation of system performance modeling data. Finally, (Bennett and Field, 2004) presents a performance engineering methodology that addresses the early stages of the development process. Is is based on UML sequence diagrams annotated with performance information using the Profile for Schedulability, Performance and Time.

These approaches for model-based software performance prediction are far from our proposal since they are conceived for internal analysis of the performance of the system and not for testing. Indeed, the main purpose of these works is not the test cases modeling as in our proposal, but the representation of the internal structure of the system and the simulation of its behavior with model analysis, to predict the performance results.

Search-based Testing for Non-functional Properties. Search-based software testing deals with the application of metaheuristic search techniques to generate software tests. In the last years these techniques have been also applied to testing non-functional properties. McMinn (McMinn, 2004) provides a comprehensive survey about the application of metaheuristics in white-box, black-box and grey-box testing. This survey also addresses non-functional testing evidencing the application of metaheuristic search techniques for checking the best case and worst case execution times of real-time systems. An extension of this survey is presented in (Afzal et al., 2009) in which the authors show that metaheuristic search techniques have been applied for testing of the execution time, quality of service, security, usability and safety. These techniques are totally different from our proposal since they are mainly based on genetic algorithms, grammatical evolution, genetic programming and swarm intelligence methods.

3 UML-TP

The Unified Modeling Language (UML) (OMG, 1997) is a widely known and applied standard notation used along the software development process. The most common and practical way to extend the expressiveness of UML is by the use of profiles. The UML Profile mechanism includes the ability to tailor the UML metamodel defining domain specific languages by means of stereotypes, tag definitions, and constraints which are applied to specific model elements. The UML-TP (OMG, 2004) (Baker et al., 2007) is the OMG standard for test modeling, implemented as a UML Profile. UML-TP is mainly used to perform functional testing whereas its application for specifying non-functional test cases is quite limited since it lacks facilities able to address specific non-functional concepts. In the next sections we first provide an overview of UML-TP and then present the limitations of the current standard version for defining test cases to measure performance and dependability properties.

3.1 Current UML-TP Standard Version

UML-TP is a lightweight extension of UML with specific concepts (stereotypes) for testing, grouped into: i) test architecture; ii) test data; iii) test behavior; and iv) test time. This extension fills the gap between system design and testing, allowing the users to have a unified model for both aspects of the development of a software product. The test architecture provides all the elements that are needed to define the test cases. Specifically, it includes the set of concepts to specify the structural aspects of the test. Among them, there are: i) the *Test Context*, which groups the *Test Cases*, and ii) the *Test Components*, which are responsible of the communication with the *SUT*.

The main constructor is the *Test Case* whose behavior can be described by sequence diagrams, state machines or activity diagrams. In UML-TP, the *Test Case* is an operation of a *Test Context* that specifies how a set of *Test Components* cooperates with the *SUT* to achieve the *Test Objective*, and to provide a *Verdict*. The behavior can be enriched with timer restrictions which are not part of the UML standard, and are included in UML-TP. Finally, another important aspect of the test specification is the test data. It is possible to model different *Data Partitions* that are obtained from a *Datapool* through *Data Selector* methods. Also, it is possible to enrich these definitions with the use of *Wildcards* and *Coding Rules*.

Generally, a UML-TP model is presented through different diagrams. Mainly, there is a UML *package diagram* representing the test architecture, and showing how the test package uses a test data package and includes the SUT model in order to allow the test elements to access the different elements under test. A *class diagram* could be used to show the structure of the test package, showing how the *Test Context* is related to the different *Test Components*, *Datapools*, and *SUT* components that are going to be exercised in the test. This *class diagram* should also model the *Test Cases* as methods of the different *Test Contexts*, representing in this way the complete test suite. There could be also a *composite structure diagram* of the *Test Context* class to show its *Test Configuration*, describing the relationships between the *SUT* and the

Test Components for this *Test Context*. Finally, each *Test Case* behavior is presented with any kind of UML *behavior diagram*, such as *State Machine Diagram*, *Sequence Diagram* or *Activity Diagram*.

UML-TP provides a mechanism of "default" behaviors, for example for the *Arbiter* and for the *Test Scheduler*. If the modeler/tester wants a different behavior for them, it is also necessary to provide an extra *behavior diagram*, mentioned in the standard as "user-defined behavior".

3.2 Limitations of the Current Version

In this section we show the main limitations of the current UML-TP standard version for modeling performance and dependability test cases. Specifically, experimenting with UML-TP for trying to model some real load testing scenarios, we realized that:

i) it does not provide support for modeling the workload concept;

ii) it does not include the most common validation facilities for performance and dependability, mainly based on the average or percentage of the response times values or the amount of pass and fail verdicts.

In load testing it is important to model the workload that is usually required from any load simulation tool. The workload defines how many operations can be concurrently executed on the SUT into an interval time. As part of testing design the tester should for instance define how the different test cases can be executed concurrently into a load testing scenario, what data they use, what is the delay between different test executions and how the test is going to reach the simulated load goal, namely the ramp-up of each test case.

As we already said, there is no workload concept defined in the UML-TP standard. However, using the UML-TP concepts provided in the standard and according to the UML-TP examples in (OMG, 2004; Baker et al., 2007), it is possible to derive a partial and tricky way to model the workload. To illustrate it, we have created an example (see Figure 1) to show how a simple workload composed by one test case ("testcase_1") executed by 150 users during a certain time, can be modeled.

In this example the *workload* is considered as a test case in the *Test Context* and the *Test Component* executes concurrently the different stimulus on the SUT for a certain time. As showed in the figure, the *Test Context* includes a test case "testcase_1", and another special test case "testcase_workload" that is in charge of the concurrent execution. The behavior of this special test case is modeled with a sequence diagram where the *Test Component* executes the "testcase_1" inside a loop. To set the total test execution time a timer "timer1" is added. The amount of users for the test case that we want to simulate is specified in a generic parameter of the sequence diagram ("maxUsers"), but it is not clear how to set it. In addition, for the distribution of the executions (think times between executions), it is necessary to store the values in a *Datapool*, and give them to the timer ("timer2") that establishes a pause in the loop after the execution of the test case. The main problem of this workload representation is that it is very tricky and incomplete since it does not allow to specify different and concurrent test cases (it is not clear how to represent more than one test case in that way) and other important concepts such as for instance the ramp-up of the test case.

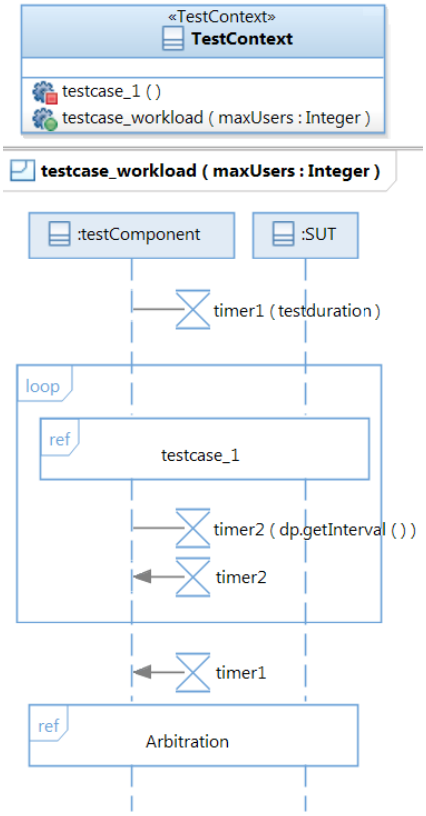

Figure 1: Workload test model from the current UML-TP specification.

Other examples presented in the standard specification of UML-TP (OMG, 2004), in order to represent the workload, include two concepts that are: the "background load", useful to generate a certain stress on the system, and the "foreground load" that includes the test cases that the user is interested to measure. These examples show that the test cases are executed in parallel, but there is no way to see clearly how

the workload is defined. Specifically, the limitations of the provided examples are: i) it is not evidenced which load should be executed against the SUT in order to verify the non-functional properties; ii) the amount of users executing the "background load" is not represented; iii) the amount of executions, the delay between executions (presented as a datapool) and the ramp-up for each test case of the workload are not clearly defined.

Another attempt to represent a workload with UML-TP is presented in (Zhu et al., 2007). In this work the authors use the UML-TP in a non-conventional way in order to model a load test by specifying in the datapool the percentage of users executing each test case. Also this workload representation is limited since the semantic of the workload representation is not in the model, it is in the way the authors interpret the content of a generic datapool.

Another important aspect that, in our opinion, is not well-covered by the UML-TP standard, is related to validation, namely how to define the verdict when a non-functional property (performance or dependability) related to a set of test cases needs to be verified. In these cases, the arbiter should be capable to express global validations in an easy way, taking into account for instance the average of the different response times of all the executions of a test case, or considering the verdicts of all the test cases executions, in order to compute the percentage of the passed test cases.

In the aforementioned book (Baker et al., 2007) and in the UML-TP standard (OMG, 2004), some examples are presented in which the global arbiter gives the verdict according to the percentage of "pass" test cases. However, it is necessary to express how to calculate this percentage in order to set the verdict. Figure 2 shows a possible representation of a user-defined behavior for an arbiter (according to the examples of the standard) asking for certain percentage of the responses to be "pass". Since this kind of validation is much common in performance and dependability testing, it is important to have a simple way to represent it.

On the other hand, and still in line with non-functional validations, the current UML-TP standard allows us to determine the minimum and maximum accepted response time values, specifying that all the response times should be within a certain range of values. We claim that these time restrictions are not enough to represent the most typical validations that usually are performed in a load simulation test, namely the average or percentage of the response times being under certain boundary.

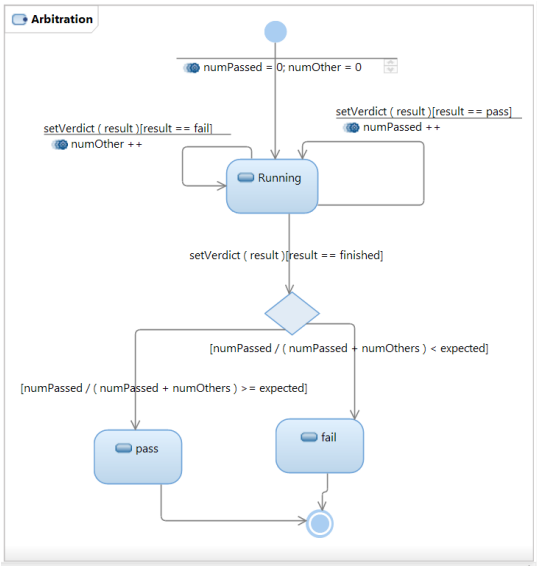

Figure 2: Arbitration to accept certain percentage of correct responses.

For both cases (performance and dependability validations) a limitation is given by the default behavior of the arbiter that has an *only-get-worse* policy. This means that if one test case reports a failure, the whole test suite fails. In our proposal we want to take into account all the test executions and provide a global verdict for the test suite according to a different policy.

4 PROPOSAL FOR EXTENDING UML-TP

In this section we present some insights for improving the expressiveness of UML-TP in the design of non-functional test cases, focusing on performance and dependability. Specifically, in the following sections we show our proposal for improving mainly two aspects: the workload specification and the verdict definition involved into non-functional validations. The proposed concepts allow for modeling a wider variety of test cases with different test goals in one single UML model. It is also important to mention that the need for extension of the UML-TP to design non-functional tests and the ideas we propose in this paper have undergone detailed and useful discussions with some members of the UML-TP development team.

4.1 Workload Information

An important lack of the UML-TP standard is the representation of the *workload*. Workload modeling is one of the most important aspects of the performance

testing activity. Designing a performance workload model very similar to the SUT environment is one of the core activities in performance testing. The test case design should define the workload including a list of parameters (load distribution, number of concurrent users, etc.) that are necessary for an accurate simulation in order to reach the test objective. Our proposal is to introduce into UML-TP standard the concept of *WORKLOAD* as a new stereotype with some tagged values in order to model all the parameters of performance testing. The stereotype would be applicable to Test Cases (to operations of a Test Context that already have a Test Case stereotype applied). With this new stereotype we can for instance specify test suites (or Test Context) with an associated workload that specifies the number of concurrent users. This information is necessary in order to verify non-functional properties.

Conceptually, the "workload" stereotype applies to the relation between the Test Context and the Test Case (see Figure 3). The different tagged values (stereotype properties) we propose to model into the workload are:

- concurrentExecutions: also known as *virtual users*, refers to the number of concurrent executions of the test case that are going to be considered in the load simulation;
- executionTime: represents the total execution time, namely for how long the test case is going to be executed;
- thinkTime: represents the delay between iterations, also known as *think times*, determines the pause between different executions of the test case for each virtual user;
- rampupTime: determines the initial time to reach the expected load, when the virtual users are increasing in the system progressively;
- startUpDelay: defines the time when the virtual users executing the test case are going to be started, relative to the beginning of the whole test suite;
- expectedDependability: represents the expected percentage of executions that should pass in order to consider the test case as passed, more details about this property are in Section 4.2.

The Test Scheduler is in charge to take into account (in its default behavior) the above parameters specified into the workload.

4.2 Non-functional Validations

With the time restrictions provided by the UML-TP standard we can only define a local verdict, namely

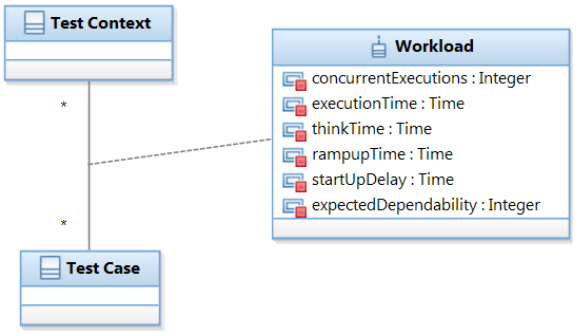

Figure 3: UML-TP extension relative to the workload concept.

if the test case takes more time than the defined restrictions, it is considered to be failed. On the other hand, the default behavior of the arbiter has the *only-get-worse* policy, what means that once it gets a fail result it never can get better than that.

Two typical situations in performance and dependability testing that are not covered by the above time restrictions and verdict definition are: i) it is needed to give the verdict according to the average of the execution times; and ii) it is needed to verify that a certain amount of the executions pass, so taking into account a policy different from the *only-get-worse* one.

In the current UML-TP standard version it is only possible to model the maximum and minimum acceptable values. Then it is not possible to report a failure according to the average or a percentage of the response times of all executions of the test case in the test context. To address this issue and improve the UML-TP time restrictions we propose the addition of new constructors that are *average* and *percentage*. The former gives the verdict considering the average of the response times, and the latter gives the verdict according to the percentage of the response times that are under the expected value.

Figure 4 shows an example of the new UML-TP Time Restrictions. In this figure "average" has a range as a parameter (0..10), and "percentage" has an extra parameter (95) indicating which percentage of the times should be in the range (we want in this case to validate that at least the 95% of the response times are in the range 0..10).

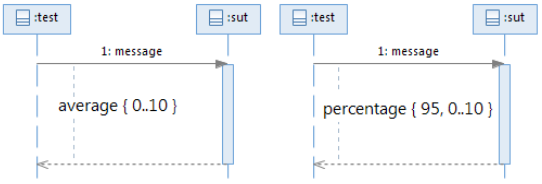

Figure 4: UML-TP Extended Time Restrictions.

On the other hand, we want to provide the tester with the capability to model that a test case is considered acceptable (with a pass verdict) if there is a certain percentage of the responses with pass verdict. For this, we consider that two changes are needed to be done on the standard profile. Firstly, we propose to add an extra property to the "Workload" stereotype, indicating the level of tolerance that we have for failures. We call this attribute "expectedDependability" referring to the expected percentage of executions that should pass in order to consider the test case as passed. However, the test execution has only one arbiter associated to the Test Context, which has the *only-get-worse* policy. We propose to add a "partial" arbiter to each test case, in order to give a verdict for each test case considering all its executions. Those arbiters should not have the *only-get-worse* behavior, instead, they should give the verdict considering the level of tolerance for the corresponding test case. Then, the main arbiter associated to the Test Context is notified of each verdict and provides the final verdict as pass only if all partial arbiters returned pass. It is important to note that the UML-TP user would only need to model the "expectedDependability", and let the default arbiter to be in charge of giving the final verdict.

From our point of view, this representation is a proposal to model the most common performance test scenarios and their typical validations. If it is necessary to calculate the verdict into another way, it is always possible to describe this behavior with another UML diagram. It is evident that the proposed extension allows to represent in a simpler way not only the same but also more information. Including these extensions into the standard UML-TP increases the expressiveness of the meta-model, consequently the resulting models are easier to develop and understand.

In the following section we present a load test modeling example to show the expressiveness of the proposed UML-TP extensions.

5 MODELING EXAMPLES WITH THE EXTENDED UML-TP

In this section we provide a load test modeling example using the extended UML-TP. We first give a short overview of the SUT and the load simulation that we want to test, and finally we show the test artifacts.

The system under test we consider as application example is an online bookstore system offering a books' catalog to the user. Basically, the system allows the users to search different books and buy them. After a market analysis and sales forecast, it has been defined that, in a peak hour, it is possible to have continuously about 500 concurrent users searching a book, while other 100 are buying a book. It was analyzed that between one execution and another one typically a user waits around 5 seconds. We want to assure that when the system is under this load situation, it is able to keep good response times and low error rates. For this, we defined the following non-functional requirements: i) the average of the duration of the search operation should be less than 15 seconds (only considering server time that is the time for having the system answer, excluding user time); ii) at least the 95% of the total amount of the buying process and the 90% of the search operation should be correctly processed by the system.

In the following subsections we present how a tester could model this common load test using the insights about UML-TP presented in this paper.

One of the proposed UML-TP extensions is about the workload. This extension allows to model in a clear and easy way the workload of the above application example. As showed in Figure 5, we model the test suite as a Test Context including two test cases, one for the search operation and one for the buying operation. To each test case a workload definition is associated including all parameters related to the users' concurrent executions. Specifically, the stereotype "Workload", includes some parameters set according to the load test definition and other ones set according to the tester experience. The former involve: i) the number of concurrent users, represented by *concurrentExecutions* value that is equal to 500 for *workload_Search* and equal to 100 for *workload_Buying* respectively; ii) the time between two different executions, named *thinkTime*, its value is equal to 5 sec for both test cases workload. The latter involve: i) the total time execution (*executionTime*), equal to one hour; ii) the way the test is going to reach the simulated load goal (*rampupTime*) that is set to 10 minutes in order to initiate the load test progressively, because it is not realistic to start 600 users at the same time; iii) the start-up time (*startupDelay*) that is set to 0 in order to start the execution of both test cases at the beginning of the load test.

In Section 4.2 we presented our idea of extending UML-TP to cope with non-functional validation and global arbiter definition. We show here how to use this extension to validate both non-functional requirements presented above.

Concerning the first requirement (the average of the duration of the search operation should be less than 15 seconds), we model the behavior of the test case "search" as showed in the sequence diagram presented in Figure 6. This model includes a time restric-

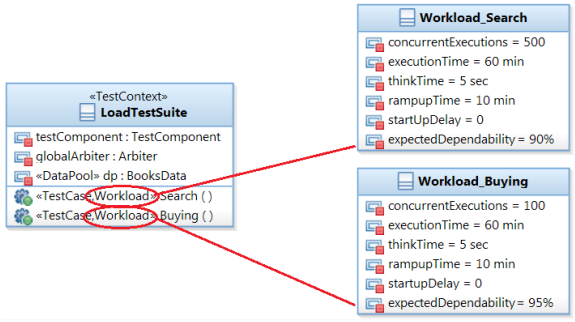

Figure 5: Test architecture with the extended workload stereotype.

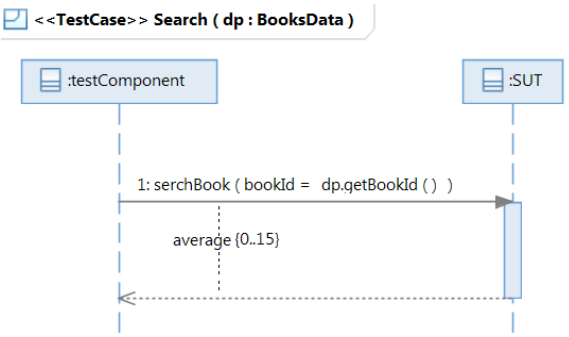

Figure 6: Test behavior of the *search* operation with the extended time restriction.

tion specifying that the average of the response times of the search operation should be less than 15 seconds. This defined time restriction allows the arbiter to give a global verdict summarizing all the response times and verifying that the average of the response times obtained from the different executions of the test case must be compliant with the expected value. By means of this global verdict it is possible to verify that the constraints expressed in the requirements' model are reached.

For validating the second non-functional requirement above defined, namely "at least the 95% of the total amount of the buying process and the 90% of the search operation should be correctly processed by the system", we indicated in the "Workload" stereotype of *Search* and *Buying* test cases what is the expected amount of correct responses (the *expectedDependability* value is set equal to 90 and 95 respectively). Then the arbiter can give a global verdict according to the specified *expectedDependability* that represents the defined tolerance to failures.

This simple application example evidences that using our insights for extending UML-TP it is possible to model in a simple and clear way the elements of a common load test scenario.

6 CONCLUSIONS AND FUTURE WORK

This paper focuses on model-based testing for non-functional requirements and specifically on the modeling languages used for defining a test model. We briefly presented the results of a systematic survey on model-driven non-functional validation that evidenced the lack of a standard and common language for designing non-functional test cases, apart from the UML-TP that per se provides only limited support for non-functional testing. We proposed in this paper some insights for extending UML-TP to represent the workload concept and the global verdict definition involved into non-functional validations. We also validated our proposal by presenting an application example of an online bookstore system, for which we defined two non-functional requirements and we showed how to apply the extended UML-TP for designing test cases for such requirements. In future work, we plan to further extend the UML-TP specification by including timing concepts of MARTE profile (OMG, 2011) and by focusing on other issues of non-functional testing such as the resource usage and the variable load during the time. In the meanwhile, we want also to include in the standard other modeling facilities specifically conceived for other kinds of non-functional testing such as stress testing or peak testing. Furthermore, we want also to investigate other research directions of model-based testing that are the derivation of an integrated test model including functional and non-functional aspects and the generation of executable test cases from this model.

ACKNOWLEDGEMENTS

This work has been partially funded by the Agencia Nacional de Investigación e Innovación (ANII, Uruguay) and by the GEODAS project (TIN2012-37493-C03-01, Spain).

The authors wish to thank the leaders of the UML-TP development team, Ina Schieferdecker and Marc-Florian Wendland, for their suggestions and useful discussions.

REFERENCES

Abbors, F., Ahmad, T., Truscan, D., and Porres, I. (2012). MBPeT: a model-based performance testing tool. In *Proc. of the Fourth International Conference on Advances in System Testing and Validation Lifecycle (VALID)*, pages 1–8.

Abbors, F., Ahmad, T., Truscan, D., and Porres, I. (2013). Model-Based Performance Testing in the Cloud Using the MBPeT Tool. In *Proc. of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE)*, pages 423–424.

Afzal, W., Torkar, R., and Feldt, R. (2009). A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976.

Apache (2001). Jmeter. http://jmeter.apache.org/.

Baker, P., Dai, Z. R., Grabowski, J., Haugen, O., Schieferdecker, I., and Williams, C. (2007). *Model-Driven Testing: Using the UML Testing Profile*. Springer-Verlag New York, Inc.

Balsamo, S., Di Marco, A., Inverardi, P., and Simeoni, M. (2004). Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5):295–310.

Bennett, A. J. and Field, A. J. (2004). Performance engineering with the UML profile for schedulability, performance and time: a case study. In *Proc. of the 12th IEEE Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS)*, pages 67–75.

Cai, Y., Grundy, J., and Hosking, J. (2004). Experiences integrating and scaling a performance test bed generator with an open source CASE tool. In *Proc. of the 19th IEEE International Conference on Automated Software Engineering (ASE)*, pages 36–45.

da Silveira, M. B., Rodrigues, E. d. M., Zorzo, A. F., Costa, L. T., Vieira, H. V., and de Oliveira, F. M. (2011). Generation of scripts for performance testing based on UML models. In *Proc. of the Twenty-Third International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 258–263.

de Oliveira, F. M., Menna, R. d. S., Vieira, H. V., and Ruiz, D. D. (2007). Performance testing from UML models with resource descriptions. In *Proc. of 1st Brazilian Workshop on Systematic and Automated Software Testing*, pages 1–8.

Dias Neto, A. C., Subramanyan, R., Vieira, M., and Travassos, G. H. (2007). A survey on model-based testing approaches: a systematic review. In *Proc. of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies (WEASELTech), in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering*, pages 31–36.

Kitchenham, B. (2004). Procedures for performing systematic reviews. In *Joint Technical Report, Keele University Technical Report TR/SE-0401 and NICTA Technical Report 0400011T.1*.

Krishnamurthy, D., Rolia, J. A., and Majumdar, S. (2006). A synthetic workload generation technique for stress testing session-based systems. *IEEE Transactions on Software Engineering*, 32(11):868–882.

McMinn, P. (2004). Search-based software test data generation: a survey: Research articles. *Softw. Test. Verif. Reliab.*, 14(2):105–156.

Mercury, H. (2001). Loadrunner. http://www8.hp.com/us/en/software-solutions/software.html?compURI=1175451.

OMG (1997). Unified Modeling Language. http://www.uml.org/.

OMG (2004). UML 2.0 Testing Profile Specification. http://utp.omg.org/.

OMG (2011). UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE). http://www.omgmarte.org/.

Pozin, B. A. and Galakhov, I. V. (2011). Models in performance testing. *Programming and Computer Software*, 37(1):15–25.

Rodríguez, F. T., Lonetti, F., Bertolino, A., Usaola, M. P., and Lamancha, B. P. (2013). Extending the Non-Functional Modeling of UML-TP. Technical Report. http://pumalab.isti.cnr.it/index.php/en/.

Smith, C. U., Llado, C. M., and Puigjaner, R. (2010). Performance Model Interchange Format (PMIF 2): A comprehensive approach to Queueing Network Model interoperability. *Performance Evaluation*, 67(7):548–568.

Utting, M. and Legeard, B. (2007). *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc.

Zhu, L., Bui, N. B., Liu, Y., and Gorton, I. (2007). MDABench: customized benchmark generation using MDA. *Journal of Systems and Software*, 80(2):265–282.