Canopus: A Domain-Specific Modeling Language for Performance Testing

Maicon Bernardino[1], **Avelino Francisco Zorzo**[1]

[1]Postgraduate Program in Computer Science – Faculty of Informatics (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 – Partenon, 90619-900 – Porto Alegre – RS – Brazil

bernardino@acm.org, avelino.zorzo@pucrs.br

Abstract. *Despite all the efforts to reduce the cost of the testing phase in software development, this is still one of the most expensive phases. In order to continue to minimize those costs, in this paper, we propose a Domain-Specific Language (DSL), built on top of MetaEdit+ language workbench, to model performance testing for Web applications. Our DSL, called Canopus, was developed in the context of a collaboration between our university and a Technology Development Laboratory from an Information Technology (IT) company. It is presented, in this paper, the overview of Canopus, including: metamodels, its domain analysis, a process that integrates Canopus to Model-Based Testing, and applied it to an industrial case study. Furthermore, we also carried out a controlled empirical experiment to evaluate the effort (time spent), when comparing Canopus with another approach widely used by industry - UML.*

Resumo. *Apesar de todos os esforços para reduzir o custo do testes de software na fase de desenvolvimento, esta ainda é uma das fases mais caras. Para continuar a minimizar esses custos, neste artigo, propõe-se uma linguagem específica de domínio (Domain-Specific Language - DSL), desenvolvida usando a ferramenta MetaEdit+, para modelar testes de desempenho para aplicações Web. A DSL, chamada Canopus, foi desenvolvida no contexto de uma colaboração entre a universidade e um laboratório de desenvolvimento de tecnologia de uma empresa de Tecnologia da Informação (TI). Apresenta-se, neste artigo, a visão geral da Canopus, incluindo: metamodelos, sua análise de domínio, um processo que integra a Canopus ao teste baseado em modelos, bem como sua aplicação a um estudo de caso industrial. Além disso, realizou-se um experimento empírico controlado para avaliar o esforço (tempo gasto), ao comparar Canopus com outra abordagem amplamente usada pela indústria - UML.*

1. Introduction

It is well-known that the testing phase is one of the most time-consuming and laborious phases of a software development process [Yang et al. 2008]. Depending on the desired level of quality for the target application, and also its complexity, the testing phase may have a high cost. Normally, defining, designing, writing and executing tests requires a large amount of resources, *e.g.* skilled human resources and supporting tools. In order to mitigate these issues, it would be relevant to define and design the test activity using a well-defined model or language and to allow the representation of the domain at a high

level of abstraction. Furthermore, it would be considerable to adopt some technique or strategy to automate the writing and execution of the tests from this test model or language. One of the most promising techniques to automate the testing process from the system models is Model-Based Testing (MBT) [Utting and Legeard 2006].

MBT provides support to automate several activities of a testing process, *e.g.* test cases and scripts generation. In addition, the adoption of an MBT approach provides other benefits, such as a better understanding on the application, its behavior and test environment, since it provides a graphical representation about the System Under Test (SUT). Although MBT is a well-defined and applied technique to automate some testing levels, it is not fully explored to test non-functional requirements of an application, *e.g.* performance testing. There are some works proposing models or languages to support the design of performance models. For instance, the SPT UML profile relies on the use of textual annotations on models, *e.g.* stereotypes and tagged values to support the modeling of performance aspects of an application. Another example is the Gatling Domain-Specific Language (DSL), which provides an environment to write textual representation of an internal DSL based on industrial needs and tied to a testing tool.

Although these models and languages are useful to support the design of performance models and also to support testing automation, there are a few limitations that restrict their integration in a testing process using an MBT approach. Despite the benefits of using an UML profile to model specific needs of the performance testing domain, its use may lead to some limitations. For instance, most of the available UML design tools do not provide support to work with a well-defined set of UML elements, which is needed when working with a restricted and specialized language. Thus, the presence of several unnecessary modeling elements may result in an error-prone and complex activity.

Most of these issues could be mitigated by the definition and implementation of a graphical and textual DSL for the performance testing domain. However, to the best of our knowledge, there is little investigation on applying DSL for the performance testing domain. Therefore, it would be relevant to develop a graphical modeling language for the performance testing domain to mitigate some of the limitations mentioned earlier. In this paper, we propose Canopus, a DSL that aims to provide a graphical and textual way to support the design of performance models, and that can be applied in a model-based performance testing approach. Hence, our DSL scope is to support the performance testing modeling activity, aggregating information about the problem domain to provide better knowledge sharing among testing teams and stakeholders, and centralizing the performance testing documentation. Moreover, our DSL will be used within an MBT context to generate performance test scripts and scenarios for third-party tools/load generators.

Consequently, the research problem is the lack of a modeling standard and/or language that aims at meet the particular needs of the performance testing domain for Web applications. This way, this study seeks to research a modeling standard for performance testing, *i.e.* to develop a DSL that meets the specific needs of the domain for modeling performance testing in a Web application, as well as its use in the MBT approach. It is highlighted that a DSL can be represented graphically, *i.e.* using graphs and diagrams, and when it is applied to the context of software testing, enables the use of the MBT approach for generating test artifacts. Hence, to formalize the exposed problem, a research question is defined to guide the research methodology.

Research Question (RQ): "*RQ0. How to improve model-based performance testing using a domain-specific language in Web applications?*"

The RQ is in line with some of the achievements, challenges, and dreams presented by Bertolino [Bertolino 2007] in the software testing research roadmap. The author asserts that both MBT and DSL are promising approaches and in actual research expansion. Together they define the dream of test-based modeling and the proposal to hold 100% automatic testing. Domain-Specific Testing (DST) is a defined term by the author as an efficient solution to allow that domain experts can express abstract specifications, *e.g.* models or languages, to be automated in their process, having as its focus on transforming the domain knowledge to improve the testing process.

From the industry perspective, the performance testing process is usually very expensive, regarding infrastructure resources and generation of test scenarios and scripts. Another significant gap to be highlighted is the lack of non-functional performance requirements in the system analysis, in addition to the absence of a standard documentation among the stakeholders about the problem domain, allowing a common understanding and interpretation of goals and aims of the system.

Regarding academia, several types of research are evolving for the purpose of automating the testing process with the aim of reducing the cost and effort applied in the activity of the generation of test scenarios and script activities, which would consequently improve the software quality. To support this challenge, one way to operationalize this process is through the MBT adoption, with the intention of sharing among stakeholders the project information regarding system behavior. Thus, it enables functional requirements and, mainly, non-functional performance requirements to be contemplated, annotated and documented into the system models on the early cycle of software development.

1.1. Objectives

The main goal of this research is to propose a domain-specific language for modeling performance testing in a Web application. To achieve the research goal, we derived the following objectives: i) Deepening the background concerning models and formalisms for performance testing; ii) Studying the models and formalisms for model-based testing; iii) Conceiving a DSL for modeling performance testing, characterized by the following representation factors of performance testing [Woodside et al. 2007]: a) Representing their features; b) Identifying their goals; c) Measuring their performance counters; d) Modeling their different user profiles as well as their behavior. iv) Validating the DSL for modeling performance testing proposed using a controlled experiment, in comparison with usual approaches for performance modeling; v) Evaluating the DSL proposed by professionals or expert groups in the field whereby the empirical study; vi) Documenting and reporting the study results, publishing them in scientific conferences, in addition to the technology and knowledge transfer to industry.

This paper is organized as follows. Section 2 describes the research methodology applied in this study. Section 3 presents the metamodels and how they are related one each other, as well as describes a model-based performance testing process. Section 4 shows the results of how we applied Canopus in a case study. Besides it, also reports the outcomes of a controlled experiment. Section 5 concludes the paper with highlights of research contributions, limitations, future directions, and points out academic publications.

2. Research Methodology

The knowledge is considered scientific if techniques that allow its verification can be applied, *i.e.*, determining the research methods that achieved such knowledge. We can define a method as a way to reach a purpose. Thus, the scientific research depends on a set of intellectual and technical procedures so that its goals are achieved, *i.e.*, research methodology. Hence, research methodology is a set of process and mental operations that had to apply in the research. In that sense, this section presents a research methodology that was planned and applied in this study developed.

This research is exploratory. Exploratory research enables to define a problem and formulate hypotheses about the topic under study [Yin 2013]. Besides, it aims to examine an issue or a non-studied problem, which has not been discussed previously by other studies. Exploratory research allows the researcher to determine a set of data collection techniques to conduct the study. In this research, we choose to use as main methods: literature review, case study, and controlled empirical experiment. Next, we present how each method will be applied in the context of the research design.

The research developed follows the objectives defined in Section 1. Thereby, we classified the nature of our proposal in applied research, with a strategy quantitative of the exploratory research type, having as its base the experimental research design. The empirical experiment method having been applied, we follow the protocol proposed by Wholin [Wohlin et al. 2012], using the instruments of quantitative and qualitative data collection. For this reason, it is an experimental research, developed in a laboratory, *i.e.*, *in vitro*. The choice to apply an empirical experiment should be in fact that the study proposal makes "how" and "why" questions, due to needs of validation of the DSL proposed to compare its performance with other approaches already used by industry.

To develop the research project proposed, we planned a research methodology organized in three phases: `Conception`, `Validation`, `Knowledge and Technology Transfer`; according to the presented research design in Figure 1. Each phase is described in details as follows:

`(1) Conception`: The first phase is divided in two blocks: `Theoretical Base` and `Development`. The former block includes the idea definition, the research question, as well as its strategy, the research design, besides a literature review in order to establish the theoretical basis to main research topics such as performance test modeling, model-based testing, and domain-specific language. This block is also responsible for defining the requirements of the DSL, and consequently, the design decisions based on these requirements. The latter block exerts the function to identify, analyze and implement the domain, in addition to studying the different frameworks and Language Workbenches (LW) [Erdweg, S. [*et al.*] 2013] for creating domain-specific languages, and also for elaborating the mechanisms to translate the visual model in a textual language, *i.e.* the generator module; `(2) Validation`: The second phase is divided in two blocks: `Utilization` and `Experiment`. The first block is responsible for applying our DSL in two Web applications: one simple and didactic, known by academic community, `TPC-W`, and the other more complex and robust, based on an industrial environment, henceforth `Changepoint`. The second block is part of the validation of a proper approach. Therefore, we intend to conduct a controlled empirical experiment with inexperienced and experienced subjects, with the purpose to plan, execute, collect data, and

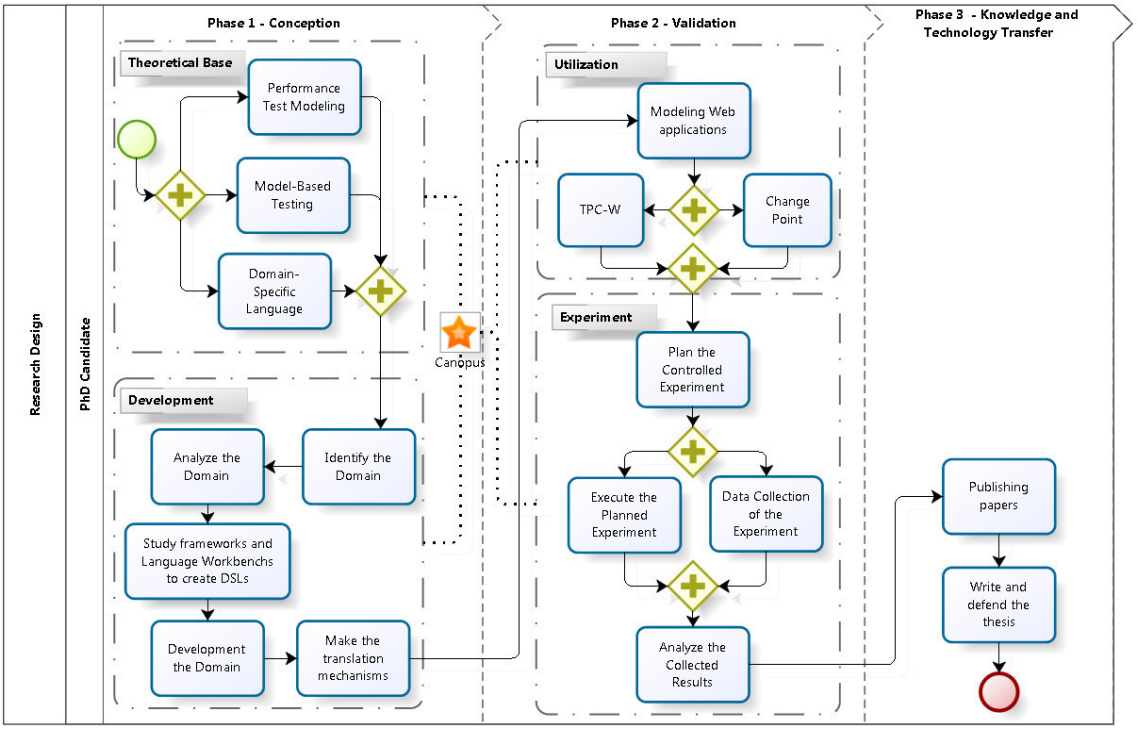

Figure 1. Research design

analyze the experiment results comparing our DSL with another approach applied by industry. Based on the enrollment context and the previously conducted research, we chose comparing our DSL with the UML approach; `(3) Knowledge and Technology Transfer`: The last phase is responsible for producing scientific essays to the academic community, as well as transferring the technology developed to our partner with the intention that it adopts our solution to improve its testing process.

Regarding data analysis, we intend to apply statistical methods such as average, median, and standard deviation, among others based on descriptive statistics proposed by Oates [Oates 2006], in order to measure the efficiency and effectiveness of the proposed DSL to answer the research questions. However, we intend to perform advanced statistical techniques to evaluate, for instance, the normal distribution and quality of our data. However, it depends on the number of experiment subjects. Table 1 presents a synthesis of the research methodology.

Table 1. Synthesis of the research methodology

Subject	Performance Testing
Topic	Performance Test Modeling
Research Question	How to improve model-based performance testing using the domain-specific language in Web applications?
Hypothesis	The performance test modeling through of a domain-specific language in Web applications can improve the quality, cost and effectiveness of performance testing.
Main Goal	To develop a domain-specific language for modeling performance testing in Web applications.

3. Canopus

In this section we present Canopus, the metamodels that compose our DSL, and describe our model-based performance testing process using Canopus to support the design of performance testing modeling of Web applications.

It is important to mention that during the design and development of our DSL, we also considered some requirements from a Technology Development Lab (hereafter referred to as TDL) of an industrial partner, in the context of a collaboration project to investigate performance testing automation. These requirements were: a) the DSL had to allow for representing the performance testing features; b) The technique for developing our DSL had to be based on Language Workbenches (LW); c) The DSL had to support a graphical representation of the performance testing features; d) The DSL had to support a textual representation; e) The DSL had to include features that illustrate performance counters (metrics, *e.g.* CPU, memory); f) The DSL had to allow the modeling of the behavior of different user profiles; g) Traceability links between graphical and textual representations should require minimal human intervention; h) The DSL had to be able to export models to specific technologies, *e.g.* HP LoadRunner, MS Visual Studio; i) The DSL had to generate model information in an eXtensible Markup Language (XML) file; j) The DSL had to represent different performance test elements in test scripts; and, k) The DSL had to allow the modeling of multiple performance test scenarios.

The rational and design decisions for our DSL are described in [Bernardino et al. 2014]. Furthermore, in that paper we discuss the performance testing domain and describe how the metamodels were structured to compose our DSL. Besides, we also present how Canopus was designed to be applied with an MBT approach to automatically generate performance test scenarios and scripts. To demonstrate how our DSL could be used in practice, we applied it draft version of Canopus throughout an exemple of use for the TPC-W e-commerce Web application.

3.1. The Language

Canopus is composed of three main parts: monitoring, scenario, and scripting.

Monitoring: the performance monitoring part is responsible for determining all servers used in the performance testing environment. For each server (*i.e.*, application, databases, or even the load generator), information on the actual testing environment has to be included, *e.g.*, IP address or host name. It is worth mentioning that even the load generator has to be described in our DSL, since we can also monitor the performance of the load generator. Sometimes, the load generator has to be split into several servers if we really want to stress the application or database server. For each host, it is possible to indicate the performance counters that will be monitored. This monitoring part requires that at least two servers have to be described: one that hosts the application (SUT) and another to generate the workload and to monitor the performance counters of the SUT.

Scenario: the performance scenario part allows setting user and workload profiles. Each user profile is associated to test scripts. If a user profile is associated with more than one test script, a percentage is attributed between the user profile and each test script, *i.e.*, it describes the percentage that that test script is executed. In addition to setting user profiles, in this part, it also is important to set one or more workload profiles. Each workload profile is composed of several elements, defined as follows: a) *Virtual users*

(VU): refers to the number of VU who will make requests to the SUT; b) *Ramp up time*: defines the time it takes for each set of ramp up users to access the SUT; c) *Ramp up users*: defines the number of VU who will access the SUT during each ramp up time interval; d) *Test duration*: refers to the total time of performance test execution for a given workload; e) *Ramp down users*: defines the number of VU who will leave the SUT on each ramp down time; f) *Ramp down time*: defines the time it takes for a given ramp down user to stop the testing.

Scripting: the performance script part represents each of the test scripts from the user profiles in the scenarios part. This part is responsible for determining the behavior of the interaction between VU and SUT. Each test script includes activities, such as transaction control or think time between activities. The same way as there is a percentage for executing a test script, which is defined in the scenarios part, each test script can also contain branches that will have a user distribution associated to each path to be executed, *i.e.*, the number of users that will follow each path.

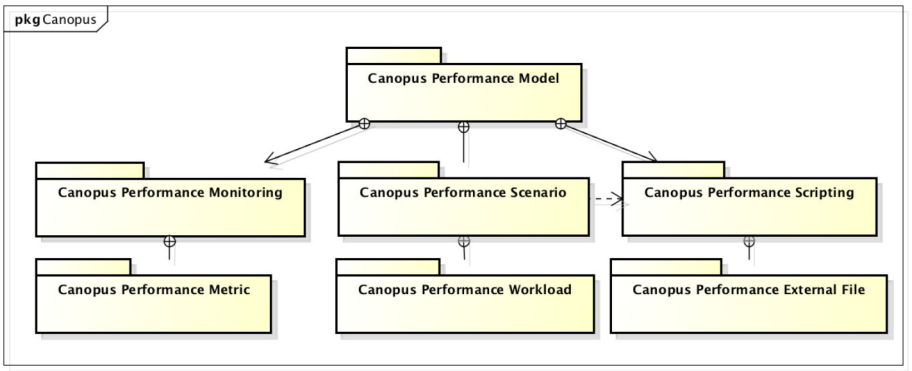

Figure 2. Canopus package diagram

To support the creation of our DSL with these parts, we chose MetaEdit+, one of the first successful commercial tools. MetaEdit+ supports the creation and evolution of each of the Graph, Object, Port, Property, Relationship and Role (GOPPRR) [Kelly and Tolvanen 2007] metatypes. Canopus has 7 metamodels represented by 7 packages (see Figure 2). The main metamodels that compose our DSL are `Canopus Performance Monitoring (CPMon)`, `Canopus Performance Scenario (CPSce)`, and `Canopus Performance Scripting (CPScr)`, which together compose the `Canopus Performance Model`. Complementing the structure, the following metamodels `Canopus Performance Metric (CPMet)`, `Canopus Performance Workload (CPWl)`, `Canopus Performance External File (CPExF)` are extended, respectively, by each one of main metamodels.

3.2. A Model-Based Performance Testing Process

The aim of our Domain-Specific Modeling (DSM) process, using Canopus, is to improve a performance testing process to take advantage of MBT. Figure 3 shows our process for modeling performance testing using Canopus. This process incorporates a set of activities that have to be performed by two different parties: `Canopus` and `Third-Party`. Besides, our DSM process is composed for seven main activities described in details next.

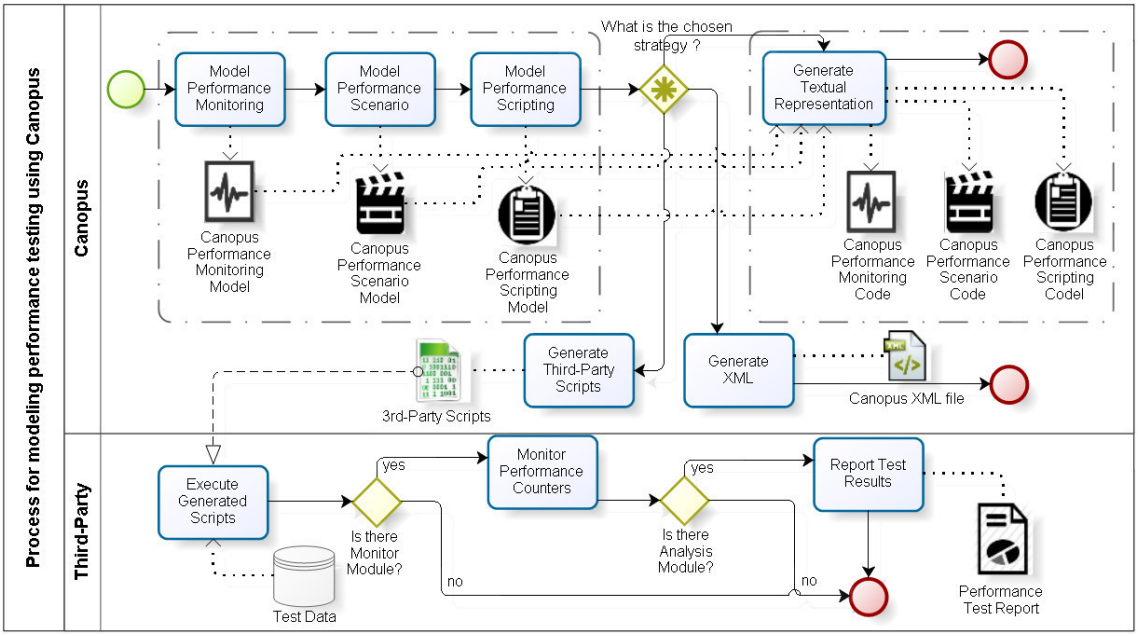

Figure 3. Model-based performance testing process using Canopus

Model Performance Monitoring: the designing of the first activity of our process is executed by the `Canopus` party. In this activity, the SUT, monitor servers and performance metrics that will be measured are defined. The milestone of this activity is the generation of a `CPMon`. This model is composed of SUT, Load Generator (LG) and Monitor objects. A Monitor object is enabled to monitor the SUT and LG objects; this object is controlled by a `CPMet` that can be associated with one or more of these objects. A `CPMet` model represents a set of predefined metrics, *e.g.*, memory or processor. Each one of them is associated with a metric counter, which in turn is linked to a criterion and a threshold.

Model Performance Scenario: the next activity of our process consists of modeling the performance test scenario. The `CPSce` model is the output of this activity. This model is composed of user profiles that represent VU that can be associated with one or more script objects. Each one of these scripts represents a functional requirement of the system from the user profile point of view. Furthermore, a script is a detailed VU profile behavior, which is decomposed into a `CPScr` model. Besides, each scenario allows modelling several workloads in a same model. A workload (`CPWl`) is constituted of setup objects of test scenario, *i.e.* ramp up, ramp down, test duration (time) and number of VU.

Model Performance Scripting: in this activity, each script object, modeled in the `CPSce` model, mimics (step-by-step) the dynamic interaction by VU with the SUT. This activity generates a `CPScr` model that is composed of several objects, such as, activity or think time. It is important to notice that two objects, *i.e.* activity and data table, can be decomposed into new sub-models. The former can be linked to a `CPScr` model that allows encapsulating a set of activities to propose its reuse into other models. The latter is associated with a `CPExF` that fills a dynamic model with external test data provided by a performance engineer. After the three first activities from our process, the performance engineers have to decide whether they generate a textual representation from the designed Canopus Performance models, an input for a third-party tool, or even a generic XML representation that might be integrated to any other tool that accepts XML as input.

Generate Textual Representation: this activity consists of generating a textual representation in a semi-natural language, a DSL-based on the Gherkin [Wynne and Hellesøy 2012] language that extends it to include performance testing information. Our design decision to deploy this feature in Canopus is to facilitate the documentation and understanding among development and testing teams.

Generate Third-Party Scripts: Canopus was designed to work also as a front-end to third-party performance tools. Therefore, even though we can generate a "generic" textual representation, our process allows integration of new features in Canopus to generate input for any testing tool. Hence, it can be integrated with different load generator tools.

Generate Canopus XML: this activity is responsible for generating a Canopus XML file. We included this feature to support the integration of Canopus with other technologies or IDEs. Hence, Canopus can export entire performance testing information from Canopus Performance models to an XML file.

Third-Party: the other three activities, shown in Figure 3, are not part of the Canopus process, depending on the third-party tool that is used, such as HP LoadRunner, MS Visual Studio or Neo Load. The `Execute Generated Scripts` activity consists of executing the performance scenarios and scripts generated for a third-party tool. During the execution of this activity the load generator consumes the test data mentioned in the data table object in `CPScr` model; `Monitor Performance Counters` activity is executed if the third-party tool has a monitoring module; and, `Report Test Results` activity is also only executed if the performance tool has an analysis module.

4. Empirical Studies Evidences

This section will present the achieved results of empirical evidences conduted during this research aiming to validate and to evaluate Canopus.

4.1. Case Study

We applied Canopus to model an application in an industrial case study, in the context of a project of collaboration between a Technology Development Lab (TDL) of Dell Computer Brazil and our university [Bernardino et al. 2016a]. Thereunto, to demonstrate how our DSL can be used in practice, we applied it throughout the Changepoint Web application, in which Canopus is used to model performance testing information supported by the model-based performance testing process. Changepoint is a commercial solution to support portfolio and investment planning, project and application management.

In short, we aim to explore two Research Questions (RQ) applying this case study:
RQ1. *How useful is it to design performance testing using a graphical DSL?*
RQ2. *How intuitive is a DSL to model a performance testing domain?*

We investigated and answered each one of the RQ based on the results of our case study and interviews conducted with a performance testing team. The performance testing team was formed by three performance engineers. Moreover, a Web-based survey was answered by fifteen performance test experts. The purpose of this survey was to evaluate the graphical elements and their representativeness to symbolize performance elements that compose each Canopus Performance metamodel (Scripting, Scenario, Monitoring).

The subjects answered a Web-based survey composed of:

(a) Statements to find whether the element is representative for a specific metamodel, based on a five points Likert scale: Disagree Completely (DC), Disagree Somewhat (DS), Neither Agree Nor Disagree (NAND), Agree Somewhat (AS) and Agree Completely (AC);
(b) Open questions to extract their opinions on each metamodel.

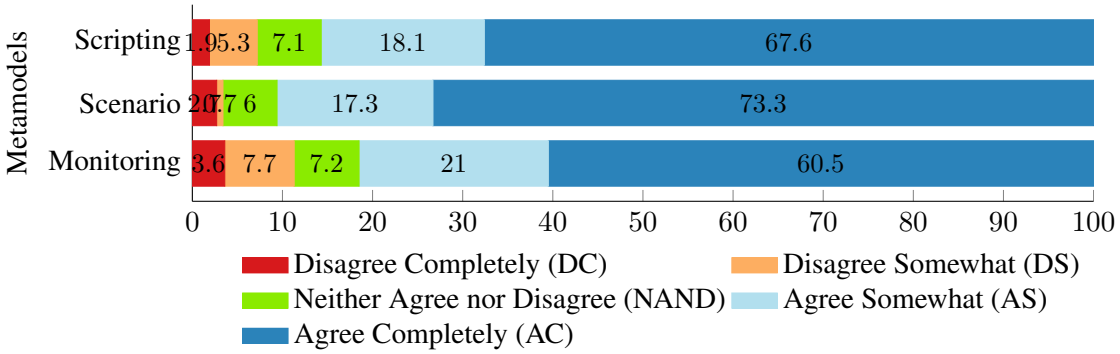

Figure 4. Frequency diagram of the graphical elements grouped by metamodel

The answers were summarized in the frequency diagram shown in Figure 4. The numbers in this figure are based on the evaluation of 37 elements: 13 elements for the `CPScr` metamodel, 10 for `CPSce` metamodel and 14 for `CPMon` metamodel. The frequency diagram presents the results grouped by each set of elements evaluated for each metamodel. As can be seen in Figure 4, 81.5% (60.5% AC + 21% AS) of the answers agree that the Monitoring elements are representative for the `CPMon` metamodel. For the `CPSce` metamodel, 90.6% (73.3% AC + 17.3% AS) agree that the elements for that metamodel are representative. Finally, 85.7% (67.6% AC + 18.1% AS) agree that the elements represent the features they intend for the `CPScr` metamodel. These results are used as part of the evaluation of Canopus.

Each of the research questions mentioned in Section 4.1, are answered next.

RQ1. The graphical representation of a notation, language or model is useful to better explain issues to non-technical stakeholders. This was also confirmed by the performance team that reported that using our approach, it is possible to start the test modeling in early phases of the development process. Furthermore, it would be possible to engage other teams during all process, mainly the Business System Analyst (BSA). The BSA is responsible to intermediate the business layer between the developer team and the product owner. Another interesting result pointed out by the subjects is that the textual representation is also very appropriate, since it allows to replace the performance testing specification. However, as expected, there is a steep curve on the understanding of the DSL notation and an initial overhead when starting using an MBT-based approach instead of a Capture and Replay (CR)-based approach.

RQ2. The case study execution indicates that the use of Canopus is quite intuitive, since the performance testing domain is represented throughout graphs, objects, relationships and properties. Visually, for instance, the scripting model can show the different flows that have been solved in several test cases on the fly, and also the decomposition and explosion features that can map objects into other graphs. This feature is also related to the reuse of partial models, characteristic of a DSL that allows to improve the productivity and to reduce the time spent on performance testing modeling activity.

4.2. Controlled Experiment

This section reports the results of a controlled empirical experiment. Thus, to support our partner company in the decision process to replace UML by a DSL, we designed and conducted an experimental study to provide evidence about the benefits and drawbacks when using UML or DSL for modeling performance testing.

We evaluate Canopus presenting an *in vitro* experiment, where the subjects analyze the design of annotated UML models and Canopus Performance models. Hence, twenty-six subjects were assigned and randomized with two groups (13 subjects per group): experienced and inexperienced. Each group started the execution of one of the two treatments (UML or DSL). This is for the purpose of evaluation with respect to the effort and suitability, from the perspective of the performance testers and engineers in the context of industry and academic environments for modeling performance testing. A complete experiment design can be found in [Bernardino et al. 2016b]. Our statistical analysis indicate that the results were valid, *i.e.*, that to design performance testing models using our DSL the effort using a DSL was lower than using UML.

To achieve our objective and purpose, we stated the following research questions:
RQ3. *What is the effort to design a performance testing model when using UML or DSL?*
RQ4. *How effective is it to design a performance testing model when using UML or DSL?*

Hence, we discuss the data collected to answer each one research questions.

RQ3. Table 2 presents the summarized effort data (time spent) to perform each task using each approach. In the table, the columns Scenario and Scripting present the average time per blocks and treatments, respectively. Based on the results summarized, the average effort using Canopus was lower than with UML in all scenarios, either to experienced or inexperienced subjects. The average time spent to design the performance testing modeling using Canopus was lower than with UML (51.08 min vs 63.69 min).

Table 2. Summarized data of the effort (minutes)

Treatments	Blocks	Blocks Average Time			Treatments Average Time		
		Scenario	Scripting	Total	Scenario	Scripting	Total
UML	Inexperienced	15.69	52.62	68.31	13.62	50.08	63.69
	Experienced	11.54	47.54	59.08			
DSL	Inexperienced	10.54	39.31	49.85	10.23	40.85	51.08
	Experienced	9.92	42.38	52.31			

Figure 5 depicts the box plot graph of the Scenario Task data set, represented by $\text{UML}_{Scenario}$ and $\text{DSL}_{Scenario}$ boxes. In the Scenario Task, the median of execution time with UML was 12.5 minutes and with Canopus it was 10 minutes. Moreover, the UML standard deviation (Std Dev) was 5.02 minutes, against 3.43 minutes for Canopus. It is important to highlight that there is one outlier in the data set for Canopus that took 17 minutes. From another point of view, Figure 5 also presents the box plot graph of the Scripting Task. This task is represented by $\text{UML}_{Scripting}$ and $\text{DSL}_{Scripting}$ boxes, where the median time to execute the UML treatment was 50.5 minutes, while for Canopus it was 39.5 minutes. Moreover, the UML Std Dev was 11.84 minutes, greater than the 7.12 minutes for Canopus. Again, notice that there is one outlier in the data set for Canopus that took 23 minutes. Figure 5 also shows the box plot graph of the summarized data set, *i.e.*, the sum of Scenario and Scripting tasks, identified by UML_{Total} and DSL_{Total} boxes.

Here, the median of execution time with UML was 62.5 minutes, inasmuch as Canopus was 48.5. It is important to highlight that the Canopus Std Dev (9.46 minutes) represents 66.8% of variation of the UML treatment (14.16 minutes). Once again, notice that there is one outlier in the data set for Canopus treatment that took 28 minutes.

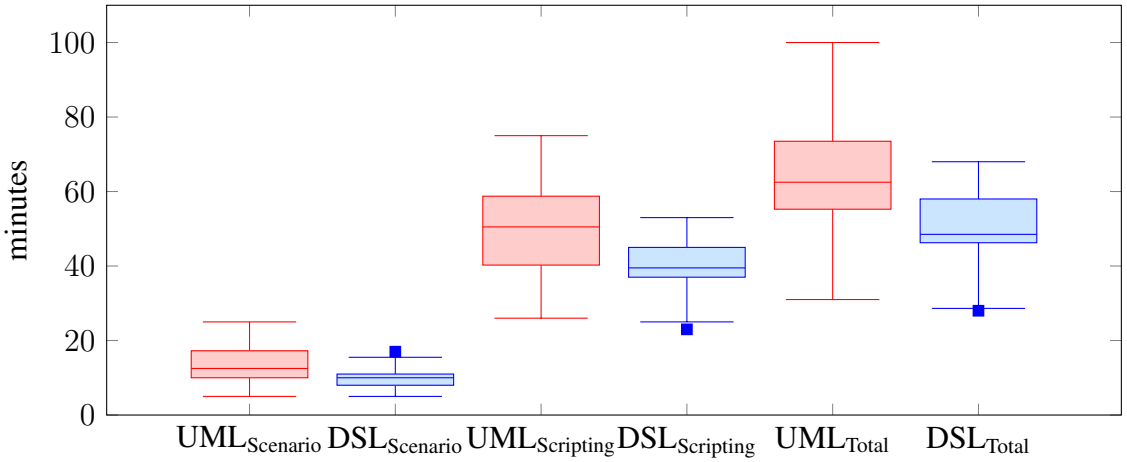

Figure 5. Box plot - treatments per task

As for hypothesis testing, we performed the Kolmogorov-Smirnov test to verify the normality of data distribution. In this context, we followed the best practice in Statistics and chose a significance level of $\alpha = 0.05$. Although, almost all results had a normal distribution, there was one exception, *i.e.*, the Scenario data set, that showed a p-value (0.355835174) greater than α. For this reason, we assumed that the distribution was not normal. Therefore, we applied a non-parametric test: Wilcoxon signed rank test. We applied a non-parametric test since it uses the median instead of average as used in parametric test, and this solves the problem with outliers. For each data set, we applied the statistical test to the paired samples, *i.e.* to compare the effort spent to model performance using UML or DSL (**RQ3**). Hence, for all samples pairs, the results of the Wilcoxon test reject the null hypothesis. Therefore, we can conclude that there is, statistically, a noticeable difference in effort to design a performance testing model when using UML and DSL. Thus, for all data sets we confirmed the alternative hypothesis.

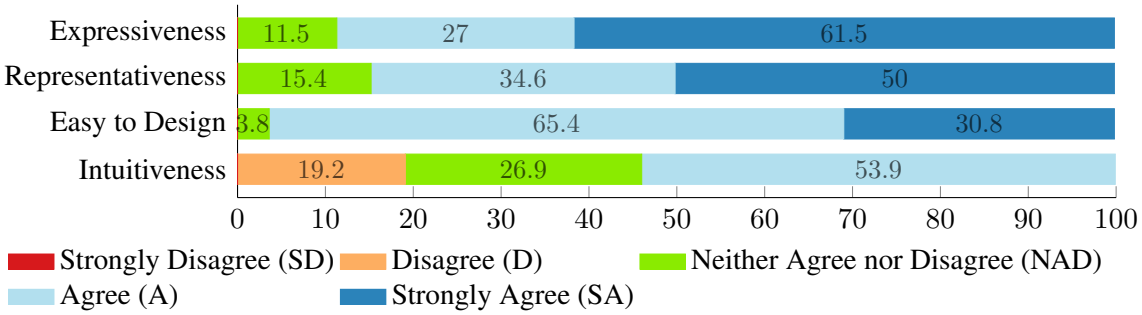

Figure 6. Frequency diagram of the Canopus

RQ4. After designing the performance models using both approaches, the subjects answered a survey composed of: (a) statements to survey how much they concur with our Canopus features; (b) open questions to extract their opinions. The answers can be seen in the frequency diagram shown in Figure 6. As can be seen in the figure,

the statement most accepted is that the Canopus has **Expressiveness** (61.5% SA, 27% A and 11.5% NAD), followed by that it has **Representativeness** (50% SA, 34.6% A, 15.4% NAD) and **Easy to Design** (30.8% SA, 65.4% A, 3.8% NAD). The statement that received the worst mark was **Intuitiveness** (53.9% A, 26.9% NAD and 19.2% D).

5. Final Remarks

The final remarks section is organized as follows. Section 5.1 revisits the achieved study contributions that support answering such a RQ. Section 5.2 summarizes the study limitations, and also sketches ongoing research and planned future work. Finally, Section 5.3 describes the academic contribution of the author in terms of publication.

5.1. Research Contributions

This section states the achievements and outcomes of this research as follows.

A pioneer graphical DSL for performance testing: we proposed, designed and developed Canopus, a graphical and textual language to model performance testing. MetaEdit+ was the LW applied to implement Canopus. Thereby, to design our textual representation, we extend the Gherkin [Wynne and Hellesøy 2012] language to include performance testing information. We chose to design both graphical and textual languages, because even though graphical may be more rich visually, not all information can be represented.

Mechanisms to easily support the integration with third-party performance tools: the MetaEdit+ have a feature that allows applying model transformation between the models instantiated from metamodels to code generation. Canopus was proposed to support dual output, graphical and textual representation. Therefore, we proposed an XML structure as another output option. The idea is applying this strategy to integrate Canopus with other load generators technologies, *e.g.* HP LoadRunner. Even so, a particular script generation for each technology as a new feature of Canopus can be also implemented.

Model-based performance testing process: our DSL was designed to integrate with an MBT approach. Thus, we propose a model-based performance testing process, which uses Canopus as a modeling tool. The process incorporates a set of activities that have to be performed by two distinct parties: Canopus and a Third-Party. Among of the set of activities, we highlighted the design, graphically, of the Canopus Performance models, as well as the following generations: textual representation, XML, and third-party scripts.

Industrial case study evaluation: we evaluated Canopus in an industrial case study. This case study applied the model-based performance testing process proposed. Thus, an experience with real-world applications - Changepoint - within an IT corporation was reported. This case study provided evidence that Canopus is feasible in real and less controlled contexts. Moreover, an analysis of a Web-based survey to evaluate the graphical elements and their representativeness, intuitiveness, and usefulness to symbolize performance elements, was answered by fifteen performance test experts.

Experimental evaluation: we conducted a controlled experiment to compare two approaches for modeling performance testing. We analyze the design of annotated UML and Canopus models for the purpose of evaluation with respect to the effort and the suitability, from the perspective of the performance testers and engineers in the context of industry and academia environments for modeling performance testing. Our findings indicate that, statistically, the effort of using a DSL was lower than that of using UML.

5.2. Limitations and Future Works

Despite this study having presented real contributions to the performance testing, MBT, and DSL areas, we identified limitations of the study contributions that can be dealt with in the future. Hence, we herein describe broader limitations to be overcome as well as opportunities for future works made during short and medium term research.

Limitations of the Canopus metamodels: the architecture of Canopus is composed of three main metamodels: monitoring, scenario, and scripting. Each one of theses metamodels requires improvements in its properties, graphical elements, or even the translation to textual representation. At first, the Monitoring metamodel needs to add new properties to some elements. Another limitation is regarding the metrics, in which the solution designed is dependent of each type of metrics instead of an abstract and generic solution. Next, the Scenario metamodel does not provide the decomposition feature among scenarios. Our industrial experience points out that this feature is necessary to express a real synthetic workload. Finally, our empirical evidences, some limitations on the Scripting metamodel were identified some graphical elements. Further investigations into strategies for designing these graphical elements can shed some light on this topic.

Canopus is not exclusive to performance testing: foremost, our intention was to design the DSL to support performance testing. Nevertheless, the proposed DSL is not restricted only to this scope. In future work, we may extend Canopus for other contexts or, who knows, to other testing paradigms, *e.g.* functional testing or security testing.

Replication of controlled experiment: in the experiment conducted, the most interesting dimensions were the effort dedicated to the subjects for performance testing modeling both approaches. However, we were aware that it was necessary to evaluate the deviation of the error rate from the models designed. Hence, we would ensure more precisely that time is spent meaningfully when compared with both approaches. Therefore, we are planning the replication of the experiment with a large sample of experiment subjects.

Comparison of Canopus and Capture & Replay (CR) techniques: we performed an experiment to compare UML and DSL models for modeling performance testing. However, for most of the industry players, MBT is not yet a standard. Therefore, for this reason, we believe that an empirical study comparing a CR-based [El Ariss et al. 2010] and DSL-based techniques require investigation and evidences.

Human-Computer Interaction (HCI) evaluation of the Canopus: although we performed two empirical studies that perform qualitative analyzes based on survey results, our approach does not apply a rigorous method supported by a widely known technique. Therefore, we already began to investigate how to conduct a heuristic evaluation for usability in HCI for user interface design of the Canopus.

5.3. Publications

During the development of this study we presented and discussed our research results in the following papers: **[Bernardino et al. 2014]**: the requirements analysis and design decisions (ICSEA); **[Bernardino et al. 2016a]**: the canopus language and an industrial case study (ICST); **[Bernardino et al. 2016b]**: an empirical experiment evaluation (SAC); **[Bernardino et al. 2017]**: a systematic mapping study on model-based testing (IET Software).

References

Bernardino, M., Rodrigues, E., and Zorzo, A. (2016a). Performance Testing Modeling: an empirical evaluation of DSL and UML-based approaches. In *31st ACM Symposium on Applied Computing*, pages 1660–1665, New York, NY, USA. ACM.

Bernardino, M., Rodrigues, E., Zorzo, A., and Marchezan, L. (2017). A Systematic Mapping Study on Model-Based Testing: Tools and Models. *IET Software*.

Bernardino, M., Zorzo, A., and Rodrigues, E. (2016b). Canopus: A Domain-Specific Language for Modeling Performance Testing. In *9th International Conference on Software Testing, Verification and Validation*, pages 157–167, Washington, DC, USA. IEEE.

Bernardino, M., Zorzo, A. F., Rodrigues, E., de Oliveira, F. M., and Saad, R. (2014). A Domain-Specific Language for Modeling Performance Testing: Requirements Analysis and Design Decisions. In *9th International Conference on Software Engineering Advances*, pages 609–614, Wilmington, DE, USA. IARIA.

Bertolino, A. (2007). Software Testing Research: Achievements, Challenges, Dreams. In *Future of Software Engineering*, pages 85–103, Washington, DC, USA. IEEE.

El Ariss, O., Xu, D., Dandey, S., Vender, B., McClean, P., and Slator, B. (2010). A Systematic Capture and Replay Strategy for Testing Complex GUI Based Java Applications. In *7th International Conference on Information Technology: New Generations*, pages 1038–1043, Washington, DC, USA. IEEE.

Erdweg, S. [*et al.*] (2013). The State of the Art in Language Workbenches. In Erwig, M., Paige, R., and Wyk, E., editors, *Software Language Engineering*, volume 8225, pages 197–217. Springer International Publishing.

Kelly, S. and Tolvanen, J.-P. (2007). *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Sons, New York, NY, USA.

Oates, B. J. (2006). *Researching Information Systems and Computing*. SAGE Publications, London, UK.

Utting, M. and Legeard, B. (2006). *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann, San Francisco, CA, USA.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., and Regnell, B. (2012). *Experimentation in Software Engineering*. Springer–Verlag, Berlin, Germany, 1st edition.

Woodside, M., Franks, G., and Petriu, D. C. (2007). The Future of Software Performance Engineering. In *Future of Software Engineering*, pages 171–187, Washington, DC, USA. IEEE.

Wynne, M. and Hellesøy, A. (2012). *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. The Pragmatic Bookshelf.

Yang, Y., He, M., Li, M., Wang, Q., and Boehm, B. (2008). Phase Distribution of Software Development Effort. In *2nd ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 61–69, New York, NY, USA. ACM.

Yin, R. (2013). *Case Study Research: Design and Methods*. SAGE Publications, London, UK, 5th edition.