PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
FACULTY OF INFORMATICS
GRADUATE PROGRAM IN COMPUTER SCIENCE

CANOPUS: A DOMAIN-SPECIFIC LANGUAGE FOR MODELING PERFORMANCE TESTING

MAICON BERNARDINO DA SILVEIRA

Thesis presented as partial requirement for obtaining the degree of Ph. D. in Computer Science at Pontifical Catholic University of Rio Grande do Sul.

Advisor: Prof. Avelino Francisco Zorzo

Porto Alegre
2016

Dados Internacionais de Catalogação na Publicação (CIP)

S587c Silveira, Maicon Bernardino
Canopus: a domain-specific language for modeling performance testing. – 2016.
175 f.

Tese (Doutorado) – Faculdade de Informática, PUCRS.
Orientador: Prof. Dr. Avelino Francisco Zorzo

1. Engenharia de Software. 2. Simulação (Computadores). 3. Informática. I. Zorzo, Avelino Francisco. II. Título.

CDD 23 ed. 005.1

Loiva Duarte Novak CRB 10/2079
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE TESE DE DOUTORADO

Tese intitulada "*Canopus: A Domain-Specific Language for Modeling Performance Testing*" apresentada por Maicon Bernardino da Silveira como parte dos requisitos para obtenção do grau de Doutor em Ciência da Computação, aprovada em 7 de março de 2016 pela Comissão Examinadora:

Prof. Dr. Avelino Francisco Zorzo– Orientador — PPGCC/PUCRS

Prof. Dr. Rafael Heitor Bordini – — PPGCC/PUCRS

Prof. Dr. Adenilso Da Silva Simão - — ICMC/USP

Profa. Dra. Silvia Regina Vergilio - — UFPR

Homologada em 16/06/2016, conforme Ata No. 012 pela Comissão Coordenadora.

Prof. Dr. Luiz Gustavo Leão Fernandes
Coordenador.

PUCRS PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Campus Central
Av. Ipiranga, 6681 – P. 32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

This thesis is dedicated
to my beloved parents,
Who educated me and enabled me
to reach at this level

"Everyone envies the beverage I drink,
but no one knows the falls I take."
(Brazilian proverb)

ACKNOWLEDGMENTS

À Deus. A força da fé me manteve firme e perseverante.

Agradeço o apoio financeiro obtido durante o período, oportunizado pela DELL, em forma de bolsa taxas. Além da oportunidade de desenvolver a minha pesquisa como bolsista vinculado ao Projeto de Desenvolvimento em Tecnologia da Informação (PDTI) no convênio DELL/PUCRS, como integrante do projeto "Center of Competence in Performance Testing".

Aos colegas do CePES e aos membros do grupo de pesquisa "CoC Perf" por terem compartilhado bons momentos durante o período do doutorado. Ao colega Elder de Macedo Rodrigues um agradecimento especial por toda a ajuda e colaboração prestada.

Ao meu orientador, professor Avelino Francisco Zorzo, por todos os ensinamentos, críticas e conselhos e, principalmente, pela confiança. Saiba que os desafios lançados colaboraram muito com o meu crescimento, tanto pessoal quanto profissional.

Agradeço aos aconselhamentos dos avaliadores, professores Adenilso da Silva Simão, Silvia Regina Vergilio e Rafael Heitor Bordini pelas considerações e sugestões de melhoria na proposta de tese e defesa da tese.

Gostaria de agradecer o apoio do professor Flávio Moreira de Oliveira pela oportunidade profissional e contribuições na publicação de artigos.

MUITO OBRIGADO aos "grandes" e mais que amigos: Elder de Macedo Rodrigues e João Batista Mossmann pelo incentivo, motivação, conselhos e ajudas. Além das cervejadas e churrascadas para manter a vida social em dia. E aos demais amigos que contribuíram de alguma forma para a realização deste trabalho.

À minha família, pelo apoio, carinho e confiança dado em todos os momentos de minha vida e, por compreenderem minha ausência nos encontros familiares durante este período.

Um agradecimento especial a minha querida mãe, a "Véia". Obrigado, do fundo do coração por sempre acreditar e confiar no meu potencial. Eu, o "Maiquinho", dedico este trabalho a você, pelo carinho e por ter me ensinado os valores e caráter que tenho hoje.

Agradecimento especial a Raissa pela paciência e por tudo que passamos juntos neste curto período de convivência. Você fez tudo parecer tão simples. Sem a tua ajuda e apoio nesta reta final, com certeza não teria conseguido.

CANOPUS: UMA LINGUAGEM ESPECÍFICA DE DOMÍNIO PARA MODELAGEM DE TESTE DE DESEMPENHO

RESUMO

Desempenho é uma qualidade fundamental de sistemas de *software*. Teste de desempenho é uma técnica capaz de revelar gargalos do sistema na escalabilidade do ambiente de produção. No entanto, na maior parte do ciclo de desenvolvimento de *software*, não se aplica este tipo de teste nos seus ciclos iniciais. Deste modo, isto resulta em um fraco processo de elicitação dos requisitos e dificuldades da equipe em integrar suas atividades ao escopo do projeto. Assim, o teste baseado em modelos é uma abordagem de teste para automatizar a geração de artefatos de teste com base em modelos. Ao fazer isto, permite melhorar a comunicação da equipe, uma vez que a informação de teste é agregada aos modelos desde as fases iniciais do processo de teste, facilitando assim sua automatização. A principal contribuição desta tese é propor uma linguagem específica de domínio (*Domain-Specific Language* - DSL) para modelagem de teste de desempenho em aplicações Web. A DSL proposta é chamada Canopus, na qual um modelo gráfico e uma linguagem semi-natural são propostos para apoiar a modelagem de desempenho e geração automática de cenários e *scripts* de teste. Além disto, apresenta-se um exemplo de uso bem como um estudo de caso realizado na indústria para demonstrar o uso da Canopus. Com base nos resultados obtidos, infere-se que a Canopus pode ser considerada uma DSL válida para modelagem do teste de desempenho. A motivação para realização deste estudo foi investigar se uma DSL para modelagem do teste de desempenho pode melhorar a qualidade, custo e eficiência do teste de desempenho. Assim, também foi realizado um experimento controlado com o objetivo de avaliar o esforço (tempo), quando comparado Canopus com outra abordagem industrial - UML. Os resultados obtidos indicam que, estatisticamente, para a modelagem de desempenho usando Canopus o esforço foi menor e melhor do que usando UML.

Palavras Chave: teste de desempenho, modelagem de desempenho, linguagem específica de domínio, modelagem específica de domínio, teste baseado em modelo, teste de *software*.

CANOPUS: A DOMAIN-SPECIFIC LANGUAGE FOR MODELING PERFORMANCE TESTING

ABSTRACT

Performance is a fundamental quality of software systems. Performance testing is a technique able to reveal system bottlenecks and/or lack of scalability of the up-and-running environment. However, usually the software development cycle does not apply this effort on the early development phases, thereby resulting in a weak elicitation process of performance requirements and difficulties for the performance team to integrate them into the project scope. Model-Based Testing (MBT) is an approach to automate the generation of test artifacts from the system models. By doing that, communication is improved among teams, given that the test information is aggregated in the system models since the early stages aiming to automate the testing process. The main contribution of this thesis is to propose a Domain-Specific Language (DSL) for modeling performance testing in Web applications. The language is called Canopus, in which a graphical model and a natural language are proposed to support performance modeling and automatic generation of test scenarios and scripts. Furthermore, this work provides an example of use and an industrial case study to demonstrate the use of Canopus. Based on the results obtained from these studies, we can infer that Canopus can be considered a valid DSL for modeling performance testing. Our motivation to perform this study was to investigate whether a DSL for modeling performance testing can improve quality, cost, and efficiency of performance testing. Therefore, we also carried out a controlled empirical experiment to evaluate the effort (time spent), when comparing Canopus with another industrial approach - UML. Our results indicate that, for performance modeling, effort using Canopus was lower than using UML. Our statistical analysis showed that the results were valid, *i.e.*, that to design performance testing models using Canopus is better than using UML.

Keywords: performance testing, performance modeling, domain-specific language, domain-specific modeling, model-based testing, software testing.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

BSA – Business System Analyst

CBMG – Customer Behavior Modeling Graph

CPM – Canopus Performance Monitoring

CPSCE – Canopus Performance Scenario

CPSCR – Canopus Performance Scripting

CPW – Canopus Performance Workload

CR – Capture and Replay

DD – Design Decision

DSL – Domain-Specific Language

DSM – Domain-Specific Modeling

DSML – Domain-Specific Modeling Language

DST – Domain-Specific Testing

EFSM – Extended Finite Machine State

EMF – Eclipse Modeling Framework

FSM – Finite Machine State

GME – Generic Modeling Environment

GMP – Graphical Modeling Project

GPL – General-Purpose Language

IEEE – Institute of Electrical and Electronics Engineers

LW – Language Workbench

LL – Lessons Learned

MARTE – Modeling and Analysis of Real-Time and Embedded Systems

MBT – Model-Based Testing

MC – Markov Chain

MDE – Model-Driven Engineering

MDD – Model-Driven Development

MDT – Model-Driven Testing

MPS – Meta Programming System

OMG – Object Management Group

OWL – Ontology Web Language

PN – Petri Nets

PLETS – Product Line of Model-based Testing tools

QN – Queueing Network

RQ – Research Question

RE – Requirement

SAN – Stochastic Automata Network

SLA – Service Level Agreement

SPE – Software Performance Engineering

SPL – Software Product Line

SPT – Schedulability, Performance and Time

SUT – System Under Test

SWEBOK – Software Engineering Body of Knowledge

TDL – Technology Development Lab

TPC-W – Transaction Processing Performance-Web

TPS – Transaction Per Second

TDD – Test Driven Development

UCML – User Community Modeling Language

UML – Unified Modeling Language

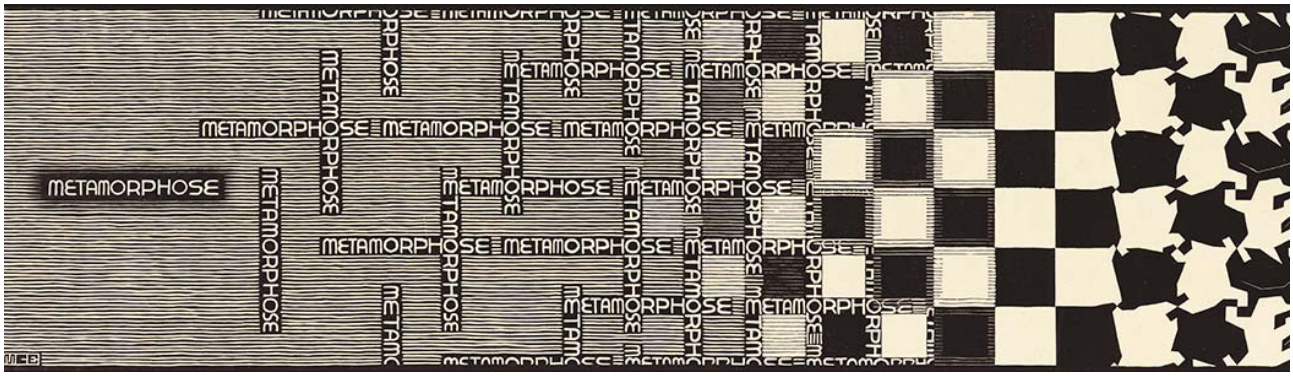
UTP – UML Testing Profile

VU – Virtual User

XML – eXtensible Markup Language

CONTENTS

1. INTRODUCTION



"For, usually and fitly, the presence of an introduction is held to imply that there is something of consequence and importance to be introduced."

— Arthur Machen

1.1 Problem Statement and Rationale for the Research

In the last years, the evolution of Web domain technologies provided a significant expansion of the number of systems and Web applications. Thus, many people are migrating from outdated client-server or stand-alone applications to some Web-based services or applications, which led application providers to invest in new technologies for infrastructure to support their applications, *e.g.* cloud computing [AFG+10] and virtualization [Vou08]. The adoption of these technologies can bring several benefits to a provider, such as dynamic provisioning of computational resources and an adaptable infrastructure. Furthermore, an infrastructure can host several isolated application servers. Moreover, these servers can have their resources, *e.g.* memory and processor time, easily expanded according to the application servers' demand [MLB+11].

Despite the benefits, the adoption of these technologies also brought about some new challenges, such as how an infrastructure provider can define a precise amount of resources for an application server, in a way that respects a predefined Service Level Agreement (SLA) [LBN02]; or, how an organization can provide a reliable service, even when it is under abnormal workload, that can be justifiably trusted by their customers [ALRL04].

One of the problems arising with the evolution of these computer infrastructures is related to achieving quality factors of developed systems hosted on these environments. In other words, one of the challenges refers to verification and validation of the quality of these systems, in particular, the performance testing validation or even to evaluate the processing limit of the computer assets to measure more accurately the SLA established in the contracts of computer service providers.

Performance testing can be applied to improve the quality of a Web-based service or application hosted on cloud computing or virtualization environments since it supports the verification and validation of performance requirements [MFB+07]. Furthermore, it also supports evaluation of the infrastructures resource consumption while the application is under different workloads, *e.g.*, to measure accurately the resources required by an application that will respect the established SLA [WB10]. Despite the fact that performance testing is a well-known technique to validate performance requirements of an application or service, there is a lack of a modeling standard and/or language to support the particular needs of the performance testing domain.

Nevertheless, there are some notations, languages, and models that can be applied to represent a system behavior, *e.g.*, Unified Modeling Language (UML) [MDN09], User Community Modeling Language (UCML) [Bar15], Customer Behavior Modeling Graph (CBMG) [MAFM99], and WebML [MFV06]. Some available modeling notations, *e.g.*, UML testing profiles [LMdG+09] and SPT UML profile [OMG05], rely on the use of textual annotations on models, *i.e.*, stereotypes and tags, to support the modeling of performance aspects of an application. The use of notations, languages or models improves the performance testing activities, *e.g.*, reducing misinterpretation and providing a standard document to stakeholders, system analysts and testers. Moreover, the use of a well-defined and concise notation, language or model can support the use of Model-Based Testing (MBT) [UL06] to generate inputs to the performance testing automation process.

MBT provides support to automate several activities of a testing process, *e.g.* test data, scenarios and scripts can be automatically generated [SKF06]. Besides, the adoption of an MBT approach provides other benefits, such as a better understanding of the application, its behavior, and test environment, since it provides a graphical representation of the System Under Test (SUT). Although MBT is a well-defined and applied technique to automate some testing levels, it is not entirely 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 Gatling [Gat15] Domain-Specific Language (DSL), which provides an environment to write the textual representation of an internal DSL based on industrial needs and tied to a particular 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 a UML profile to model specific needs of the performance testing domain, its use presents some limitations: (a) Most of the available UML design tools does not provide support to work with only those UML elements needed for a specialized language. Thus, the presence of unused and unrequired elements may result in an error-prone and complex activity; (b) The OMG defines UML diagrams restricted to their semantics. Therefore, in some cases, the available UML elements and their semantics can restrict or even prevent the modeling of some performance features of the Web domain. For instance, in some situations, it can be required that another test activity make use of a response data from previous

test activities. This situation is frequent in an enterprise scenario, but it is not easily represented using a UML profile.

It is important to highlight that UML is useful to analyze and design the architecture and the behavior of a system [HGB08]. Furthermore, it is a standard notation that does not imply an implementation decision; besides, it is helpful for representing higher level concepts and the initial domain glossary. When compared to UML, Domain-Specific Languages (DSL) [Fow10] are less general and are based on an implementation strategy. That is, UML is used at an independent implementation level, whereas DSL is used at a dependent implementation level. DSL are restricted languages that can be used to model concepts directly in a particular problem domain. These languages can be textual, like most programming languages, or graphical. Furthermore, each DSL is a domain-specific code generator that maps domain-specific models into the required code.

Most of these issues could be mitigated by the definition and implementation of a graphical and textual DSL to the performance testing domain. However, to the best of our knowledge, there is little investigation on applying DSL to the performance testing domain. For instance, the Gatling DSL provides only a textual representation, based on the Scala language [Gat15], which ties to a particular load generator technology - it is a script-oriented DSL. The absence of a graphical representation could be an obstacle to its adoption by those performance analysts that already use some graphical notation to represent the testing infrastructure or the SUT. Furthermore, as already stated, the use of a graphical notation provides a better understanding of the testing activities and SUT to the testing team as well as to developers, business analysts, and non-technical stakeholders. Another limitation is that the Gatling DSL binds to a particular workload solution.

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 thesis, we propose Canopus, which aims to provide a graphical and textual DSL 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, Canopus 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 thesis seeks to study and 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: "*How to improve model-based performance testing using a domain-specific language in Web applications?*"

The research question is in line with some of the achievements, challenges, and dreams presented by Bertolino [Ber07] 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.

Despite the fact that performance testing is an active research field, studies investigate how to apply MBT approaches to automate the performance testing activities essentially started to be reported in the last decade, and they are still in its early stages [SKF06] [KSF10] [dSdMRZ+11] [CCO+12]. Furthermore, the lack of a standard to represent performance testing information is one of the major challenges for both academic and industrial practitioners.

From the industry perspective, the performance testing process is usually very expensive, regarding infrastructure resources and generation of test scenarios and scripts [AT10]. 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 product 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 of the early cycle of software development.

In this context, despite performance testing being a recent field due to distribution and diversification of the infrastructure (*e.g.* cloud computing [AFG+10] and virtualization [Vou08]) and technology evolution, (*e.g.* Web application, Web services) many research studies were developed over the last decade [RSSP04] [KRM06] [LW08] [AATP12]. The majority is centered on proper approaches to generate and execute performance testing under the perspective of Capture and Replay (CR) technique [EAXD+10], *a.k.a.* Record and Playback. However, as is known to happen in other fields, such as Model-Driven Development (MDD) [HT06], Test Driven Development (TDD) [Bec02], Model-Driven Testing (MDT) [BDG+07], and Behaviour-Driven Development (BDD -an evolution of TDD) [WH12] this has been a little-explored gap. With the aim of answering these questions this study seeks to design models using DSL and apply the MBT approach to automation of performance testing in Web applications.

This thesis investigates this gap, and it is characterized by the growing concern about applications' performance in detriment of optimization of the infrastructure resources consumption. In other words, given greater emphasis when the environment is on the cloud or delegated for third-party providers. Thereunto, we seek to aggregate knowledge with the purpose of identifying and designing them into a DSL for modeling performance testing in Web applications.

The Web application is the scope of this thesis given our know-how of the research project developed in the industrial software projects, tools and models studied and tested in a research project. The scope holds to the premise that the application should run under the network protocol HTTP (HyperText Transfer Protocol), which falls into client-server architecture [Ser95], multi-tier architecture [Eck95], among others. This constraint appears due to the load generators - performance testing tools - often implementing their solutions based on protocols intrinsic to each application.

The DSL proposed is not restricted only to this scope. Given the incremental development methodology to the creation of DSL. Future work may extend Canopus for other contexts or perhaps even other testing paradigms.

1.2 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:

- Deepening the background concerning models and formalisms for performance testing;
- Studying the models and formalisms for model-based testing;
- Conceiving a DSL for modeling performance testing, characterized by the following representation factors [WFP07]:
  - Representing the features of performance testing;
  - Identifying the goals of performance testing;
  - Highlighting the performance counters measured by performance testing;
  - Modeling the different user profiles and their behavior.
- Validating the DSL for modeling performance testing proposed using a controlled experiment, in comparison with techniques, methods or usual approaches for performance modeling;
- Evaluating the DSL proposed and analyzing the representation power of the performance test features by professionals or expert groups in the field whereby the controlled experiment, case study and/or survey was conducted;

- Documenting and reporting the study results, publishing them in scientific conferences, in addition to the technology and knowledge transfer to industry.

With the aim of answer the research question and to attend the goals outlined, this thesis presents Canopus, which proposes the following contributions:

- Analyzing the domain of performance testing for automation of their testing process;
- Development of a domain-specific language for modeling performance testing;
- Implementation of a modeling tool that supports the graphical model of Canopus;
- Definition of an abstract structure that represents the natural language of Canopus;
- Translation and mapping of graphical model of Canopus to different technologies of load generators;
- Creation of an approach for automatic generation of test scenarios and scripts based on models using domain-specific language in Web applications;
- Generation of empirical evidence of use and application of the DSL and MBT in the performance testing context in a Web application.

1.3 Summary of Contributions and Thesis Outline

Chapter 2 brings an overview of the background that supports the main topics investigated in this thesis. Initially, foundations of performance testing and its main area are described. In addition to this, information is presented concerning its types, a process to design and execute performance testing, a variety of testing tools that support the performance testing automation. It also introduces some notations, languages and models for modeling performance testing. Next, we extend the discussion about performance modeling. Hence, we present highlights about those notations, languages, and models that served as inspiration for our ideas to propose our DSL. We present an overview about formal and semi-formal models used to model performance testing. The former is concerning early-cycle predictive model-based performance approach. The latter is regarding late-cycle measurement-based performance approach. Next, we define our DSL as well as tools to develop one. Finally, we provide an overview of the state-of-the-art regarding the use or the application of MBT or DSL approaches in the performance testing context.

Chapter 3 presents the research methodology planned and applied in this study. We characterize our research based on scientific methods. We divided this chapter into two sections as follows: in the first we describe our research design, presenting how each method was applied in the context of the research evolution; in the second we introduce the research setting, contextualizing the environment of the research group, as well as, our industrial partner.

The contributions of this thesis are described in Chapters 4, 5, and 6. Chapter 4 discusses the analysis of the performance testing domain and presents the language and the metamodels that compose our DSL, as well as the requirements and design decisions. It is important to mention that during the design and development of Canopus we also considered some requirements from an industrial partner, in the context of a collaboration project to investigate performance testing automation [BZR+14]. We also show an example of use to demonstrate the use of Canopus to support the design of performance testing modeling of a Web application.

Chapter 5 applies Canopus to model an application in an industrial case study [BRZ16]. Thereunto, to demonstrate how our DSL can be used in practice, we applied it throughout an actual case study from the industry, in the context of a project of collaboration between a Technology Development Lab (TDL) of Dell Computer Brazil and our university. Firstly, we describe our model-based performance testing process using Canopus. For this purpose, we present an industrial case study to evaluate the applicability of Canopus to model performance testing. 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?*

Chapter 6 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. In Chapter 6, we validate Canopus presenting an *in vitro* experiment, where the subjects analyze the design of annotated UML models and Canopus Performance models. 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 [BZR16]. Our results indicate that, for performance modeling, the effort using a DSL was lower than using UML. Our statistical analysis showed that the results were valid, *i.e.*, that to design performance testing models using our DSL is better than using UML. To achieve our objective and purpose, we stated the following Research Questions (RQ):

RQ3. *What is the effort to design a performance testing model when using UML or Canopus?*

RQ4. *How effective is it to design a performance testing model when using UML or Canopus?*

RQ5. *How intuitive/easy is it to design a performance testing model when using UML or Canopus?*

Finally, Chapter 7 concludes the thesis, revisiting the achieved thesis contributions, describing the study limitations, sketching ongoing research and future directions, and summarizing the academic contribution of the author of this thesis during his academic career.

2. BACKGROUND

"The more you learn, the more you have a framework that the knowledge fits into."

— Bill Gates

2.1 Overview

The background represents an important research step [Yin13]. In order to base the concepts and definitions about the areas of knowledge and unify the understanding of study topics related to this thesis, the following sections present the theoretical foundation. Section 2.2 introduces performance testing. Section 2.3 presents languages and notations used to model performance testing. Section 2.4 defines Domain-Specific Language (DSL) as well as the tools to develop one. Finally, Section 2.5 provides an overview of the state-of-the-art regarding the use or the application of an MBT or DSL approach in the performance testing context.

2.2 Performance Testing

"I just want people to focus on the performance."

— Janet McTeer

Performance engineering is an essential activity of Software Performance Engineering (SPE) [Smi02]. It's main focus is to improve scalability and performance, revealing bottlenecks in the SUT. SPE provides support for the evaluation of systems' performance through two distinct approaches: an early-cycle predictive model-based approach and a late-cycle measurement-based approach [WFP07]. In this context, we are interested in a late-cycle measurement-based approach, since in our approach the generated performance testing scripts and scenarios are applied only when the system has already been developed, and can therefore be executed and measured. However, our approach is not limited

to late-cycle, since our purpose is the engagement of the performance team from early cycles of the development software process.

The predictive-based approach, unlike the measurement-based approach, can be created and analyzed without the software being necessarily developed, so its application is commonly performed in the early stages of the software development process. The predictive-based approach describes how the system operations use the computational resources, and how the limitation and concurrence of these resources affect such operations. Among the models used in predictive-based approach, we can identify the following types: Queueing Networks (QN), Petri Nets (PN), Stochastic Automata Network (SAN), among others [WFP07]. As for the measurement-based approach, it has the aim of modeling user behavior and workload in order to run the SUT to collect performance metrics and later analysis of the results. Both approaches have their advantages. While performance testing based on predictions does not need the infrastructure for analysis, performance testing based on measurements produces more accurate results, but at a cost of time-consuming measurements [BDMIS04]. Given its accuracy, a way to guide the implementation of the SPE is by conducting performance testing based on measurements of software.

In this regard, SPE is a set of activities and heuristics to ensure that the performance analysis effort is used throughout the software development lifecycle. The starting point for a performance analysis starts from the analysis of non-functional requirements for the performance evaluation of performance metrics (*e.g.*, response time, throughput), and the expected results of performance testing [Smi02]. It is noteworthy that the non-functional requirements can be classified in other classes, not being restricted to performance, such as security, scalability, reliability, etc. [ZG07]. The non-functional requirements can be conceptualized as attributes or restrictions on a system, because they refer to "how well" the software does something, unlike the functional requirements in that their focus is to determine "what" the software does [CPL09].

In summary, a measurement-based approach supports the performance testing activity. Performance testing aims to identify potential bottlenecks or performance shortfalls, determining the system processing limits, and verify that the performance of software meets the specified performance non-functional requirements [WFP07]. For instance, it can be used to determine the application under test's behaviour under different conditions, such as insufficient memory or low disk space, performing testing over part or the whole of the system, under normal and/or stress load. Hence, performance testing can be classified roughly into two categories [MFB+07]:

(1) Load Testing: The test focuses on determining or evaluating the behavior of a system under normal workload. Mainly, to verify that the system meets the specified performance non-functional requirements;

(2) Stress Testing: The test aims to determine the behavior of a system when it performs beyond the normal workload. Further, this test reveals the failure points when the system is submitted to huge workloads.

Molyneaux [Mol09] asserts that performance testing can be classified as load testing, stress testing and soak (or stability) testing. This new category proposed by the author intends to test the SUT over an extended period of time (endurance), in which the aim is to reveal problems only when submitted in this category. Each one of these differs from each other based on their workload and the time that is available to perform the test. They can also be applied to different application domains such as desktop, mobile, Web service, and Web Application.

According to Meier *et al.* [MFB+07], performance testing determines and validates the speed characteristics, reliability and/or scalability of a system under a given workload. This type of testing focuses on measuring the responsiveness of the system (response time), network throughput, and utilization levels of computing resources (CPU, memory, disk, etc.) to meet the performance objectives for a given system. Among its main objectives is the definition of system performance features; identification underperformance of the system, *i.e.* bottlenecks; suggesting the system optimization points; estimate an appropriate hardware configuration to the SUT; among others. The process of designing and executing performance testing is composed by a set of well-defined activities [MFB+07]:

(1) *Identify the test environment*. This activity includes the definition of the physical environment for testing and production, including the resources and tools available to the performance testing team. Here, the physical environment comprises hardware, software, and network setup. It becomes an essential activity for a good understanding of the test environment results in a more efficient planning and test project, supporting the identification of testing challenges early in the project. In some cases, this process has to be reviewed periodically throughout the project's life cycle;

(2) *Identify performance acceptance criteria*. This activity aims to identify the objectives, assumptions and constraints about response time, throughput, and use of computing resources. *A priori*, the response time is a user concern, while throughput is a business problem, and resource utilization is a concern for the system level;

(3) *Plan and design tests*. This activity aims to identify key scenarios, determine variability among representative users, and simulate that variability, define test data and establish performance metrics as well as collect them.

(4) *Configure the test environment*. This activity is responsible for preparing the test environment and the tools and resources needed to perform the test. In some cases, it is important to make sure that the test environment is instrumented to ensure the monitoring of computer resources.

(5) *Implement the test design*. Development of test scripts and scenarios of planned performance testing.

(6) *Execute the tests*. Implementation of performance monitoring and running the test. In addition to validating the tests, test data, and results.

(7) *Analyze results reports and retest*. Responsible for summarizing, consolidating and sharing the test results. If monitored metrics are within acceptable limits and all desired data are collected, the test is terminated for a particular scenario and a specific infrastructure configuration.

Based on this, in order to automate some of these activities of performance testing, a variety of performance testing tools has been developed to support and automate performance testing activities. The majority of these tools take advantage of one of two techniques: Capture and Replay (CR) [EAXD+10] or Model-Based Testing (MBT) [UL06]. In a CR-based approach, a performance engineer must run the tests manually one time on the Web application to be tested, using a tool on a "capture" mode, and then run the load generator to perform the "replay" mode. There are several tools for this purpose, to name a few: Apache JMeter [JLH+10], HP LoadRunner [Hew15], IBM Rational Performance Tester [CDD+08], Borland SilkPerformer [BCM07], Microsoft Visual Studio [Mic15b], and Neo Load [Neo15]. In an MBT approach, a performance engineer designs the test models, annotates them with performance information and then uses a tool to automatically generate a set of performance test scripts and scenarios. There are some notations, languages and models that can be applied to design performance testing, *e.g.* Unified Modeling Language (UML) [OMG15], User Community Modeling Language (UCML) [Bar15] and Customer Behavior Modeling Graph (CBMG) [MAFM99]. Some of these notations are only applicable during the designing of the performance testing, to support its documentation. For instance, UCML is a graphical notation to model performance testing, but it does not provide any metamodels able to instantiate models to support testing automation. Nevertheless, other notations can be applied later to automate the generation and execution of performance test scripts and scenarios during the writing and test execution phases. Another way to design testing for a specific domain is using a DSL.

In this context, the next section extends the discussion about performance modeling and highlights the models that support the performance modeling, which were the inspiration for our ideas to propose our DSL.

2.3 Performance Modeling

With the evolution of SPE [Smi02] [WFP07], numerous types of models can be used to performance testing. These models, when used for such a test scope, intend to characterize non-functional performance requirements that the system must meet. Thus, this section presents some of the different techniques, methods and notations for modeling performance testing, assessing the particular features of each model so that they may be automated with the MBT approach.

In this context, different formalisms are used to model the performance testing. These models, when used in this test scope, are intended to formalize the non-functional performance requirements that the system must meet. Besides, the formal models can also be used to provide automation through the MBT approach. Finite State Machines (FSM) [Cho78], Markov

Chains [Bea78], Stochastic Automata Networks [PA91], Queueing Networks [Cha94] and Petri Nets [Mol82] are examples of formal models susceptible to integration with the MBT approach.

Another perspective of performance modeling is semi-formal, models that are commonly used in industrial environments, *e.g.*, User Community Modeling Language (UCML) [Bar15], the UML profile for SPT (Schedulability, Performance and Time) [OMG05] and the UML profile for MARTE (Modeling and Analysis of Real-Time and Embedded Systems) [OMG09]. The models of UML profiles [Sel07] are annotated with stereotypes and labels to write down information related to performance testing, *e.g.*, number of virtual users, application/server paths, user behavior and/or workload information.

Creating visual representations of the workload model that both, developers and testers, can understand intuitively is not a simple abstraction. Typically, this is accomplished by focusing on the perspective of iterations of users, not system activities. This workload model should contain all data needed to create the test case or test script, and in turn, generate the workload represented by the model [Bar04]. One of the alternative documentation and visualization methods that represents the data workload and meets these requirements is known as the UCML [Bar15] model, popularly used for performance testing consultants.

The UCML model is a visual notation for modeling the workload distribution of an application to performance testing. UCML has a graphical language that consists of symbols that allow the representation of user profiles, sets of activities that the user interacts with in the system, operations and transactions in the application, as well as the frequency rates by which these operations can be performed [Bar15]. However, this model has some limitations to the performance test, such as representation of the monitoring elements, *e.g.* SUT and performance metrics; definition of Service Level Agreement (SLA) [LBN02] (*e.g.* Response Time,) not to mention the transactions that should be monitored.

Figure 2.1 presents a UCML model example developed to represent the variability of different user profiles of a library system. The profiles are divided into `New User`, `Member`, `Administrator`, and `Vendor`. All of them perform some activities in common, which are represented by the black flows, while the specific activities are colored according to their user profile.

The traditional UML diagrams [OMG15] allow their extensibility so that they can be applied in different domains. Various types of performance models can be generated from a set of scenarios depicting the typical response of a system and the consumption of computational resources. This performance specification can be represented in a UML model with notes, using the UML profile for SPT [OMG05] [GK09]. The application of the model should be annotated with performance stereotypes, to assess the scalability of the application in scenarios with higher demand and competition. Some examples of applications that make use of these stereotypes are shown in [RVZG10] [TWESAO10] [dOMVR07] [BDH06] [PW04].

The main objective of stereotypes is to help the test team in two activities: improving application performance; and, providing information to the model that allows the automation of the generation of performance test scripts and scenarios.

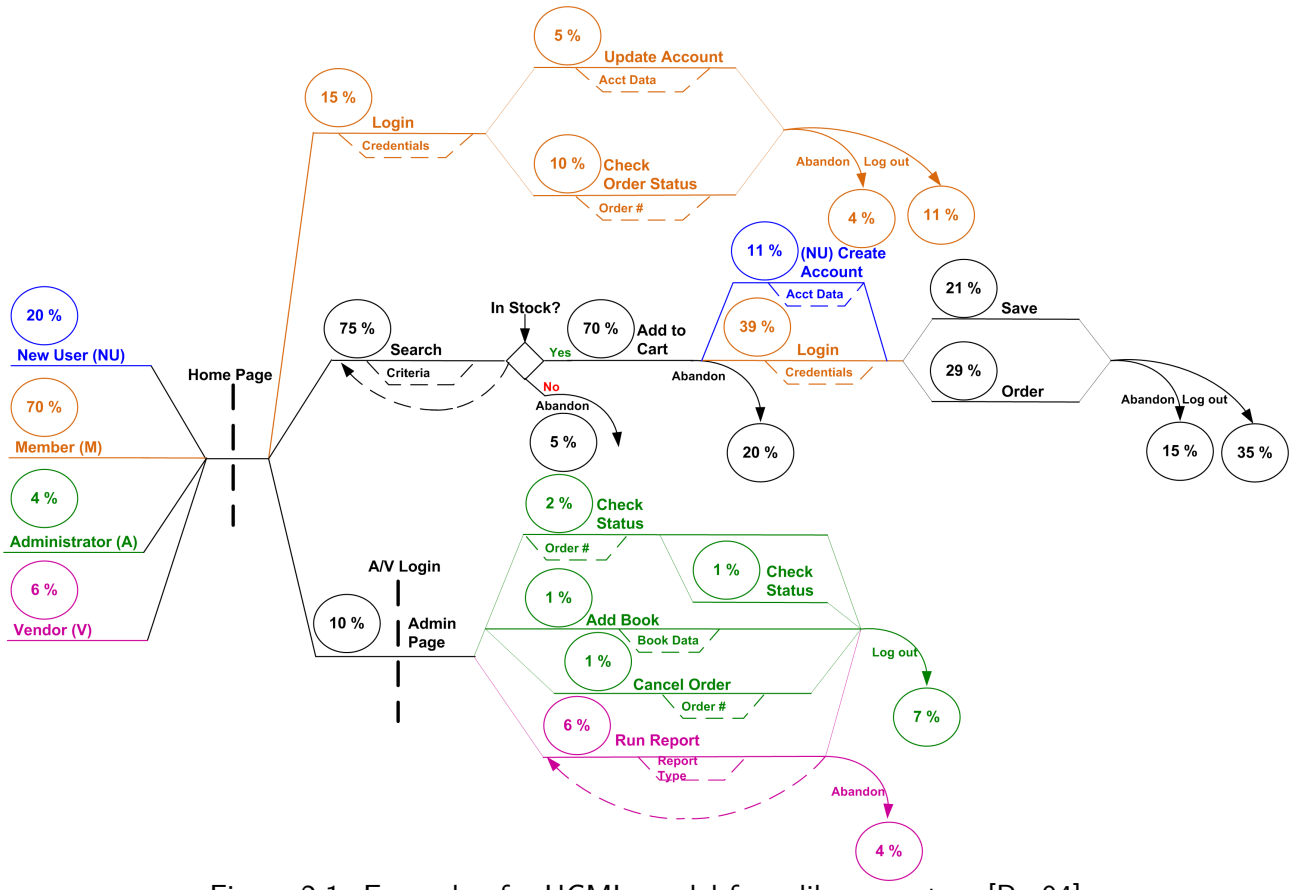


Figure 2.1: Example of a UCML model for a library system [Bar04]

Following the same idea as SPT, the UML profile MARTE [OMG09] is a notation for the Model-Driven Development (MDD) on Real-Time and Embedded Systems (RTES) promoted by OMG. This UML profile supports the specification, design, verification and validation stages of RTES systems. It aims to replace its precursor, the UML profile for SPT, to meet the demand for quantitative aspects of time and express the non-functional properties and execution platforms. All these are special concerns of designers and developers of RTES [DTA+08].

MARTE notation consists of setting foundations for the description of the model based on RTES. These basic concepts are refined for modeling and analysis of problems, providing necessary support for the specification of RTES features from specification to detailed design. In this context, the intention is not to define new techniques for RTES, but to support them. Thus, the UML MARTE profile provides facilities so that the models can incorporate necessary information to perform a specific analysis such as performance and scalability. In addition to defining a generic framework for quantitative analysis with the intention of refining and/or specializing any other type of analysis, MARTE notation also provides a general purpose mechanism for expressing Non-Functional Properties (NFP) in UML models (*i.e.* performance requirements, such as memory size, throughput or energy consumption), and to specify restrictions on these properties. This mechanism is the ability to annotate UML diagrams with these stereotypical constraints.

At last, Customer Behavior Model Graphs (CBMG) [MAFM99] [MDA04] is a state transition graph that can describe a sequence of requests. A sequence of requests defines a session, whereas a set of sessions represents a workload of a Web application. The CBMG is capable of representing every session of a Web Application. Together, these sessions are clustered by an algoritm

to obtain patterns of user interaction, *i.e.* users that have similar behavior in the navigation. The graph is composed by nodes that represent each one of the states, and transitions between these states are represented by arrows that bind one element with another. These transitions assign a probability, which represents the likelihood that a user profile will perform a request. A CBMG is designed for each user profile identified, since each one of them has different transition probabilities.

Figure 2.2 presents a CBMG model for an occasional buyer of a Web store. This graph represents the customers that navigate in the Web Store looking for existing products, such as new books or itineraries for travel, but most of the time end up not buying. This CBMG is composed of seven states (node symbol). The flow starts in the `Entry` state. Each state has one or more transition flows to reach other states or the special state `Exit`. Note that the CBMG of Figure 2.2 is just a simple example. The complexity of the model depends on the robustness of the Web application. Therefore, it can have many other states and transitions.

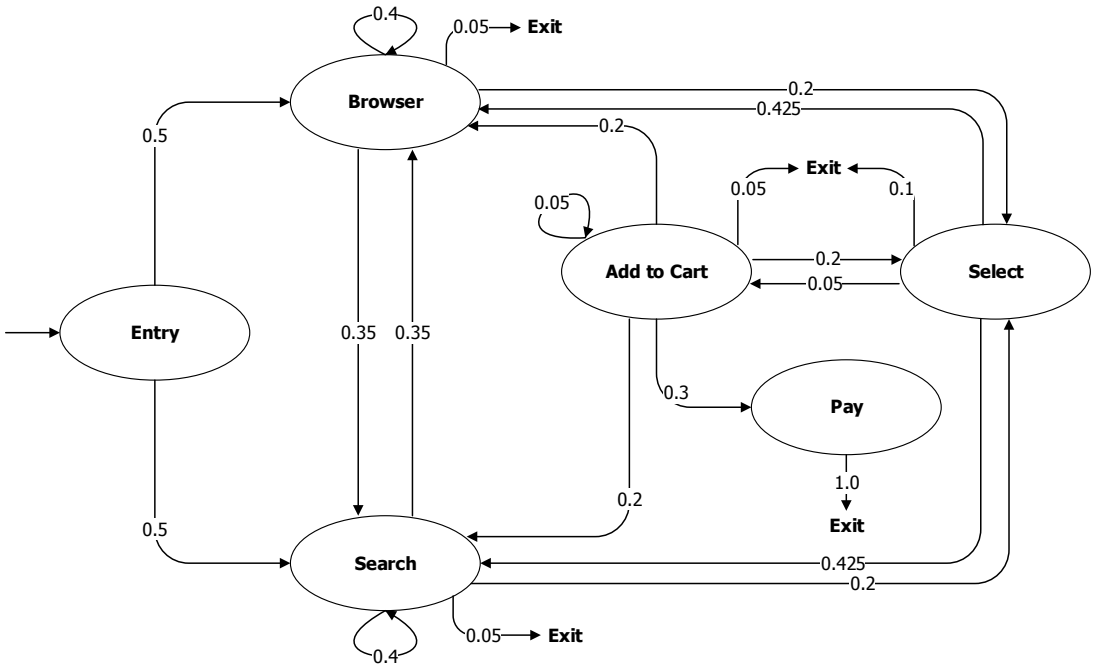


Figure 2.2: Example of a CBMG model for an occasional buyer of a Web store [MAFM99]

2.4 Domain-Specific Language

> *"Domain-specific language (noun): a computer programming language of limited expressiveness focused on a particular domain."*
>
> — Martin Fowler

Software Engineering through Software Reuse [Kru92] [FK05] seeks to establish standards and processes for software reuse in order to consider ways to promote a process able to identify, organize and group common functionalities into a single application domain. Hence, it is possible

to compose a collection of applications with a specific set of features that enable the construction of reusable software components including, for example, models.

The performance testing process, as well as the development of software, generally produces a set of artifacts that allows its reuse in applications with similar objectives. Through these reusable artifacts, it can increase productivity, improve reliability and reduce costs related to the software development process. In the context of reuse, one can determine a set of concepts, rules and procedures common to a group of systems with similar objectives. Thus, when grouped, a set of computer applications that share relevant features, such as requirements, objectives, standards and procedures, obtains the domain of these applications.

One of the methods available in the literature to model and identify the features of an application domain is Domain Engineering [WL99] [PBvdL05], which is a key concept in systematic software reuse that in turn has the domain as its key idea. This area's main purpose is to enable the same domain systems to be built from the same process and artifacts, enabling the reuse of common concepts and features [PDA91]. The Domain Engineering process permits that, from the execution of their activities, results are obtained as design patterns, code generators, reference architectures, or the Domain-Specific Language (DSL) [Sof10], also known as Domain-Specific Modeling Language (DSML), or just Domain-Specific Modeling (DSM) [KT07].

DSL, also called application-oriented, special purpose or specialized languages, are languages that provide concepts and notations tailored for a particular domain [Fow10]. Nevertheless, to develop a DSL, a DSM is required to lead to a solid body of knowledge about the domain. DSM is a methodology to design and develop systems based on Model-Driven Engineering (MDE) [Sch06]. One important activity from a DSM is the domain analysis phase, which deals with a domain's rules, features, concepts and properties that must be identified and defined. Hence, domain-specific modeling languages as well as transformation engines and generators are technologies that combined provide support to develop MDE. MDE is an emerging and promising approach that uses domain modeling in different levels of abstraction for developing and testing systems increasing their automation to code generation or model interpretation [BCW12].

From a domain model, one can obtain a DSL aimed at developing particular problems of a modeled domain, having the main advantages of creating rules and documents, features and important properties of the domain. Currently, a strategy to support the creation and maintenance of a DSL activity is to adopt metamodeling tools. Such tools are called Language Workbenches (LW) [Fow09]. There are some available LW, such as Microsoft Visual Studio Visualization and Modeling SDK [Mic15a], Meta Programming System (MPS) [Jet15], Eclipse Modeling Framework (EMF) [EMF15], Generic Modeling Environment (GME) [GME15], Xtext [The15], and MetaEdit+ [Met15]. There are LW that allow the definition of a DSL through textual definitions in similar formats to a Backus-Naur Form (BNF) [Bac78]. Moreover, others enable the use of graphical diagrams for defining a DSL addition of templates for code generation.

Besides implementing the analysis and code generation process, an LW also provides a better editing experience for DSL developers, being able to create DSL editors with power similar to

modern IDE [Fow10]. Thereby, DSL are classified with regard to their creation techniques/design as: internal, external, or based on LW [Gho11].

Internal DSL are often created on top of an underlying programing language and are embedded within the language that implements it. Therefore, it is also known as embedded DSL [Gho11]. Usually, internal DSL are built with support from General-Purpose Languages (GPL), such as Lisp, Smalltalk, Ruby, Python, and Boo.

Conversely, external DSL are independent of a programming language. Therefore, it can be compiled, interpreted or executed in accordance with its implementation. External DSL have their own syntax or in some cases can be represented by a predefined language, *e.g.* XML (hence, it's *a.k.a.* stand-alone DSL). However, they need a parser to process it. Moreover, they allow the developer to create a new syntactic, semantic, and linguistic infrastructure for the problem domain [Gho11].

In turn, DSL based on LW technique/design are used to express information in a concise and more abstract way, providing better understanding and visualization of the problem domain. Furthermore, LW provide support for the use of a graphical representation of the DSL. Thus, DSL that implement a graphical representation provide a better communication mechanism, *e.g.* a DSL user can work with visual elements instead of textual ones. Moreover, unlike the use of internal and external DSL, graphic-based DSL are easier to use and also provide a better understanding of the application domain [Fow10].

Therefore, the use of DSL presents some advantages, such as [vDKV00] [Gho11] [GFC+08]:

(a) Better expressiveness in domain rules, allowing the user to express the solution at a high level of abstraction. Consequently, domain experts can understand, validate, modify and/or develop their solutions;

(b) Improving the communication and collaboration between testing and development teams, as well as non-technical stakeholders;

(c) Supporting artifact and knowledge reuse. Inasmuch as DSL can retain the domain knowledge, the adoption of DSL allows the reuse of the preserved domain knowledge by a mass of users, including those inexperienced in a particular problem domain;

(d) Enabling better Return On Investment (ROI) in the medium- or long-term than traditional modeling, despite the high investment to design and deploy a DSL.

Despite that, the use of DSL can present some disadvantages, such as [vDKV00] [Gho11]:

(a) The high cost to design and maintain a DSL, especially if the project has a moderate or high complexity;

(b) The high cost for training DSL users, *i.e.* steep learning curve;

(c) Difficulty to define an adequate scope for a DSL. It is necessary to understand the problem domain and involve experts in order to ensure an appropriate abstraction level and language expressiveness;

(d) A company becoming dependent on an in-house language that is not applied anywhere else. This can make it difficult to find new staff to their team, as well as to keep up technological changes;

(e) In case of executable DSL there are issues related to the performance loss when compared to source code written by a developer.

2.4.1 DSL Design Methodology

The methodology proposed by Bierhoff [Bie06] offers an incremental approach for creating DSL. This methodology consists in building a DSL from incremental research of a number of existing applications in a domain. The main advantages of the incremental model of building DSL are observed primarily in the reduction in costs associated with the DSL creation project, which are initially lower. The boundaries of the area and, therefore, also the DSL are defined in the process, and you can still make use of much earlier DSL, since their use may occur according to the advancement of the creation process. As another approach, proposed by Deursen [vDKV00], developing a DSL comprises the following steps:

- Analysis: Based on the definition of the scope, the relevant knowledge in this domain has to be collected and recorded, appropriatelly, in order to translate this knowledge into semantic elements or notations as well as possible operations in a computer language. Furthermore, the analysis step is responsible for designing the DSL to describe domain applications;
- Implementation: Must ensure the construction of a library that implements semantic elements or notations that were documented in the previous step. In the library, this step requires the development of a compiler to translate programs written in the DSL into a sequence of library calls.
- Use: Consists in using the DSL to build the desired application in the domain.

2.5 Related Work

> *"Research is what I'm doing when I don't know what I'm doing."*
>
> — Wernher von Braun

Our DSL relies on work from several fields. In this chapter, we briefly described some essential concepts that underlie our research. The purpose of this chapter is to provide an overview

of the state-of-the-art regarding the use or the application of model-based performance testing. Thus, comparisons and syntheses of the related work considered relevant in the literature have been analyzed. The adopted inclusion criterion was that a work had to define or use models, notations, approaches, techniques, methodologies or tools in performance testing. There is a lack of novel models, languages and supporting tools to improve the performance testing process, mainly in the automation of scenarios and script generation. To the best of our knowledge, there is little work describing performance testing tools [RSSP04] [KSF10] [AATP12].

Ruffo *et al.* [RSSP04] developed their research at the University of Turin in Italy. The authors propose the solution called WALTy (Web Application Load-based Testing), which is an integrated toolset with the objective of analyzing the performance of Web applications through a scalable *what-if* analysis. This approach is based on a workload characterization being derived with information extracted from log files. The workload is modeled using Customer Behavior Model Graphs (CBMG) [MAFM99]. WALTy generates synthetic workload objectively based on the characterization of the user profile. Thus, it applies the following steps in its methodology: (a) To identify the case study and logical analysis of the Web application; (b) To identify the logic states of the Web application and map the Web objects for each logical state (HTML pages, Web scripts, etc.); (c) To analyze the physical infrastructure (servers, network bandwidth, etc.); (d) Automatically generate the CBMG models from log files; (e) To set the test plan, *i.e.*, replicate the Web application to a dedicated test environment; (f) To generate synthetic workload through virtual users based on CBMG generated models; (g) To collect monitored and reported performance metrics. This approach focuses in the final phase of the development process, since the system must be developed to collect the log application and then generate the CBMG. Conversely, our approach aims to support the performance requirements identification, and it is performed in the initial phase of the development process to facilitate the communication among project stakeholders.

Krishnamurthy *et al.* [KSF10], from the University of Calgary in Canada, present an approach that uses application models that capture dependencies among the requests for the Web application domain. This approach uses Extended Finite State Machines (EFSM) [LY96], in which additional elements are introduced to address the dependencies among requests and input test data. The approach is composed of a set of tools for model-based performance testing. One tool that integrates this set of tools is SWAT (Session-Based Web Application Tester) [KRM06], which in turn uses the `httperf` tool - an open source request generator - to submit the synthetic workload over the SUT. From the standpoint of performance modeling, it is worth noting that the approach implements two models: a workload model and an application model, which are transformed into synthetic workload models to be submitted to `httperf`. To validate the approach, the study presents two case studies using systems based on open-source sessions to evaluate the performance of the applications and determine whether the proposed approach is robust e§nough to model different types of Web applications. The first is a pet store based on Java technology, and the second is a bidding system called RUBiS (Rice University Bidding System). However, this approach presents some disadvantages, such as being restricted to generating workload for a specific workload tool and

the absence of a model to graphically design the performance testing, *i.e.* a model to determine the performance metrics for which monitoring is desired for each server that composes the SUT infrastructure. This is one of the requirements contemplated by Canopus. In turn, Canopus aims to be a modeling language independent of technology, responsible for generating performance test scenarios and scripts for different workload generators, *e.g.* HP LoadRunner or MS Visual Studio.

Abbors *et al.* [AATP12], from Åbo Akademi University in Finland, present a model-based performance testing tool, henceforth MBPeT. This has a measurement technique based on Master-Slave architecture through which the performance of Web applications and Web services is measured. The tool uses Probabilistic Timed Automata (PTA) to model the virtual user behavior and then to generate workload to the system in real-time. The master server of MBPeT is responsible for analyzing (Parser) the PTA model as input; next, it generates abstract outputs that will serve as input to the slaves; finally it generates the test report aggregating the results of the obtained metrics. The slaves, in turn, have the main function of generating the workload (Load Generator), and monitoring the resources of the SUT, reporting the metrics collected by the master server and realizing requests (Adapter) to be submitted to the SUT. The workload generator approach can be used in two ways to run performance testing: for a certain number of concurrent virtual users; or for certain Key Performance Indicators (KPI), *e.g.*, response time. Like our approach, the MBPeT is a measurement-based approach. Nonetheless, this approach, proposed by Abbors *et al.*, does not include, in its model, the aspects of performance requirements, *e.g.* response time, for automatic generation of SLA, such as proposed by Canopus.

Pozin and Galakhov [PG11] describe a study for the organization and execution of automated performance testing for the information systems class. The approach is based on the proposed metamodel as tools for describing artifacts' properties of workload experiments. The proposal is divided into four metamodels: requirements, system, load, and measurement. The interrelationship among the models used to a particular technology in order to apply the concept of the model-based performance testing involves the following steps: (a) Determination of testing objectives; (b) Development of testing program and procedures; (c) Test preparation; (d) Load generation; (e) Data collection; (f) Interpretation and analysis of results.

The proposed approach uses, as a solution for generating workload, IBM Rational Robot testing tools. The technology covers all aspects of planning, load experience and analysis of its results. This is similar to the solution proposed in Canopus, where language is divided into three metamodels: scenario, monitoring, and scripting, which compared with the approach of Pozin and Galakhov, would be compatible, respectively, with the models: load, measurement and system. However, large comparisons between Canopus and the proposed approach by the authors could not be performed since the study does not provide details about the formalism used to design performance testing models, nor what technology is used to generate workload. The study presents no empirical evidence to demonstrate step by step the use of the proposed approach, although it has several projects analyzed to develop and test the proposed approach.

Lutteroth and Weber [LW08], from the University of Auckland in New Zealand, present a study to model realistic workload for performance testing. They argue that predictive models such as stochastic models are easier to represent, besides allowing greater coverage of the different system iteration flows. This analysis is justified, since the load models are limited to reproduce, usually, the user behavior iterations through sessions recorded with the use of technologies such as Capture and Replay [EAXD+10], *a.k.a.* Record and Playback. The proposal of the authors is to further specifications to software-oriented forms, which are represented graphically by flowcharts. Among the annotated features needed in the models for generating workload, stand out: (a) Stochastic iteration of users: the possibility of making stochastic behavior dependent on the session history; (b) Generation of form parameters stochastically and recognition of returned pages; (c) Suitable parameters for the workload. The authors assert that the proposed methodology is independent of technology and can extend its application to other system types based on request-response system classes, which fall into Web applications. The approach of Lutteroth and Weber aims to just automate the generation of test scenarios and scripts through workload models and user iteration behavior with the SUT. On the other hand, the Canopus model has a charge for determining performance metrics for computational resources utilization, enabling the systematic generation of SLA. However, it is noteworthy that the analysis of stochastic models presented in this study shows evidence that predictive models combined with measurement models (scenarios and scripting) can generate promising results for performance testing.

Although these works introduce relevant contribution to the domain, none of them proposed a specific language or modeling notation to design performance testing. There are a few studies investigating the development of DSL to support performance testing [BZGL07] [Spa12] [Gat15]. Bui *et al.* [BZGL07] propose DSLBench for benchmark generation. Spafford [Spa12] presents the ASPEN DSL for performance modeling in a predictive-based approach [WFP07]. Meanwhile, Gatling DSL [Gat15] is an internal DSL focused in supporting a measurement-based approach. Differently from them, Canopus provides a graphical and textual DSL to design performance testing in a measurement-based approach.

Bui Ngoc Bao *et al.* [BZGL07] from New South Wales University in Australia used the approach based on Domain-Specific Modeling (DSM) to develop a new DSL, entitled DSLBench, for the benchmark generation domain. The DSM approach has some advantages over other benchmark generation frameworks. It provides a simple modeling environment that reduces the learning curve since it directly uses concepts abstracted from the performance testing domain. Furthermore, DSLBench creates a complete load test suite, able to perform comprehensive load testing including data population, generating random data, processing of test cases and collecting performance metrics. The proposal associated with code generation allows the generation of a benchmark application from a high-level model for performance testing. DSLBench is implemented using Microsoft DSL Toolkit with MS Visual Studio 2005 Team Suite as a plug-in to support the performance test design capacity. The article presents a case study based on applications developed with the .NET framework using the C#language. Canopus aims to be a DSL for modeling performance testing

of Web applications as well; based on the best of the acquired knowledge, there is no language that can support performance testing for Web applications in their entirety. However, DSLBench proved to be the most similar to the approach proposed by Canopus. Despite their similarity, one of the major disadvantages of DSLBench is to be integrated with the .NET platform, which becomes totally dependent on proprietary technologies while Canopus aims to become a solution for modeling performance testing of Web applications independent of the performance testing tool.

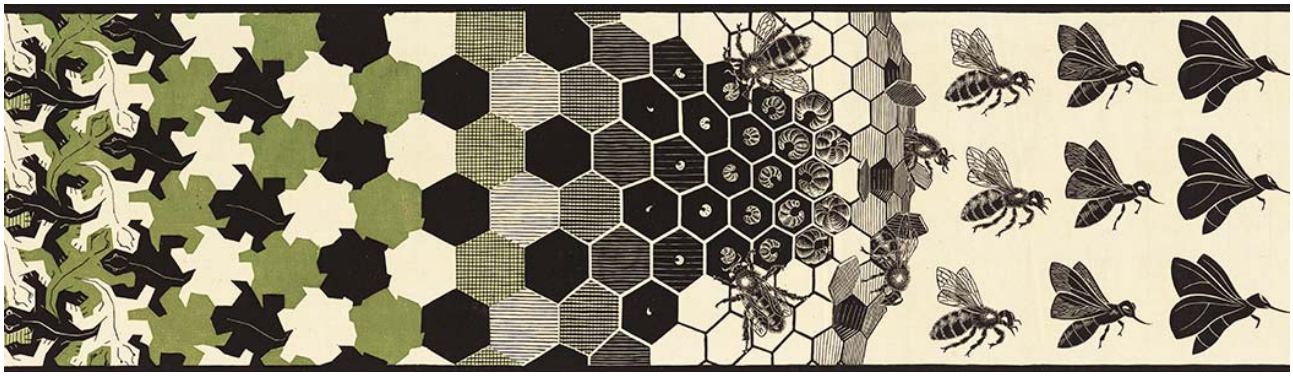
Spafford and Vetter [Spa12] present a DSL for performance modeling, entitled Abstract Scalable Performance Engineering Notation (ASPEN), a scalable abstract notation for performance engineering. The purpose of the notation includes a formal specification of an application performance behavior and an abstract machine model. Unlike Canopus, models through which performance can be compared with other techniques, in which the focus is on measuring models, ASPEN differs in considering predictive models through which performance can be compared with other techniques, such as analytical models (*e.g.* Matlab), hardware emulation (*e.g.* RAMP or DEEP), among others. The study presents two practical examples of the application of the proposed language: a three-dimensional model of Fourier (3D Fast Fourier Transform) and an application of molecular dynamics. The authors also propose a set of tools that allows scientists to balance their applications and systems on factors such as speed and accuracy.

On the DSL development perspective, different solutions for creating DSL have been proposed, also known as Language Workbenches, including both commercial and open-source. The two most popular commercial products today are MetaEdit+ from MetaCase [KLR96], and Visual Studio Visualization and Modeling SDK from Microsoft [Mic15a]. The open-source arena is no different. Although a plethora of solutions exist, ranging from solutions like Generic Modeling Environment (GME) [GME15] and Meta Programming System (MPS) [Jet15] from JetBrains, to Eclipse plug-ins as Eclipse Modeling Framework (EMF) [EMF15], Graphical Modeling Project (GMP) [GMP15], and Xtext [The15]. A comprehensive list of existing tools with the analysis of their features is presented in [LJJ07]. Moreover, a systematic mapping study on DSL is summarized in [dNVN+12].

2.6 Chapter Summary

This chapter introduced the key concepts associated with performance testing and domain-specific language, and also compared the proposed research with related studies. Given the complexity and cost involved in performance testing, the aim of this study is to investigate how a DSL can contribute to the quality, cost and effectiveness of performance testing. Thereunto, Chapter 3 will present the research methodology, research design and research context regarding the environment in which this study was developed.

3. RESEARCH METHODOLOGY



"To achieve great things, two things are needed: a plan, and not quite enough time."

— Leonard Bernstein

3.1 Overview

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 chapter presents a research methodology that we plan and apply in the study developed in this thesis.

This research is exploratory. Exploratory research enables to define a problem and formulate hypotheses about the topic under study [Yin13]. Besides, it aims to examine an issue or a non-studied problem, which has not been discussed previously by other studies [SCL91]. 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 empirical experiment. Next, Section 3.2 we present how each method will be applied in the context of the research design. Section 3.3 presents the research context, describing the environment of research group as well our industrial partner.

3.2 Research Design

"If we knew what it was we were doing, it would not be called research, would it?"

— Albert Einstein

The research developed in the thesis follows the objectives defined in Chapter 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 [WRH+12], 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 thesis 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 3.1. Each phase is described in details as follows:

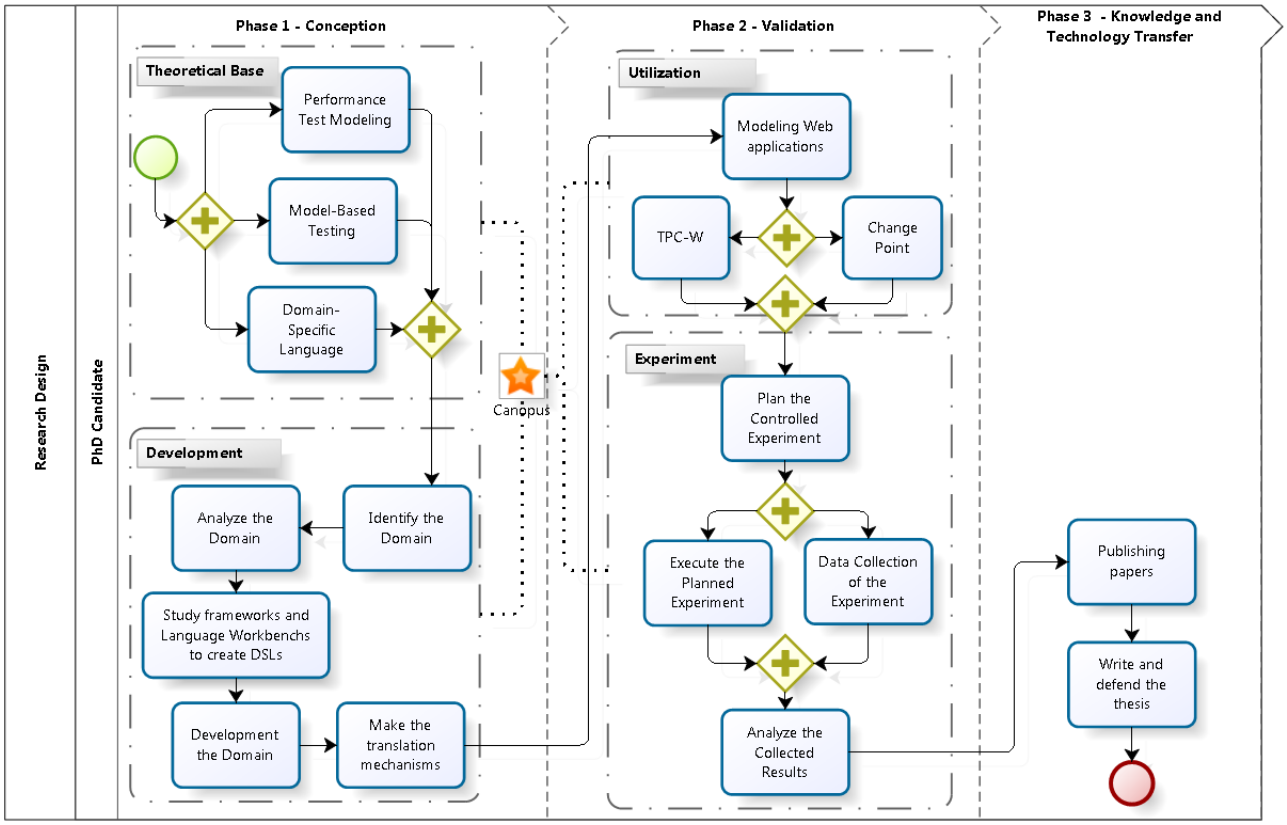


Figure 3.1: Research design

(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) [ESV+13] 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` [Men02], and the other more complex and robust, based on an industrial environment, henceforth `Changepoint` [Cha15]. 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 analyze the experiment results comparing our DSL with another approach applied by industry. Based on the context enrollment and the previously conducted research, we chose comparing our DSL with the UML [OMG15] approach;

(3) `Knowledge and Technology Transfer`: The last phase is responsible for producing scientific essays to the academic community, as well as transfering 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 [Oat06], 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 3.1 presents a synthesis of the research methodology.

Table 3.1: Synthesis of the thesis

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.

3.3 Research Context

This study was conducted in cooperation with the Technology Development Lab (hereafter referred as TDL) of Dell Computer Brazil, which is a global IT company. It is a global IT company whose development and testing teams are located in different regions worldwide to develop and test in-house solutions in order to attend their own demand systems on a global scale of sales of computer assets. The aim of this cooperation is to experiment and develop new strategies and approaches for software testing.

The software process adopted by TDL is briefly described as follows. The development team implements the unit tests, which performs the preliminary tests in a dedicated development environment. After a stable version has its requirements properly attended, a version is published in the test environment for the test team to perform the functional and integration testing. If the version presents failures, the development team refactors the application and reinitializes the process. If not, the version is implanted in a performance environment to perform the load and stress testing. Some applications depend on the complexity of the production environment and the volume of infrastructure applied, given the high-level costs to maintain the replicated environment [AT10]. Lately, since the application attends to quality requirements established, *e.g.* performance, a software publisher is responsible for deploying the version in the production environment.

The TDL develops and tests their solutions in different platforms, languages and Integrated Development Environments (IDE), which includes, respectively, Windows, Linux; Java, C#, C++, Phyton, Ruby, PHP, ASP; Eclipse, NetBeans, Visual Studio, RubyOnRails; besides making use of commercial solutions such as Siebel, PeopleSoft, Changepoint and Oracle, in which they integrate with their legacy systems.

The performance testing teams use commercial tools, frameworks and open-source to automate their activities partially. However, often due to the complexity of test, the performance testing teams need to learn a new technology, tool or framework to allow the testing of non-trivial applications. This way, the performance testing team constantly has a steep learning curve, due the complexity of the technologies and tools used to carry out their activities.

In a controlled experiment [RdOC+15] previously performed with the TDL team, we observed that many professionals have little knowledge or familiarity with modeling, even though these notations, such as UML, are commonly used by the industry. Thus, adopting and training a team to work becomes the testing process and their teams become more homogeneous. Another factor observed, due to the global teams distribution, in development, test or infrastructure, the performance testing teams do not follow the standard process to execute their performance testing activities, resulting in redundancy, low reuse, rework, and expensive costs, etc.

We propose a pilot study in order to mitigate or eliminate these factors, by creating a domain-specific language for modeling performance testing to adopt the MBT approach by the partner through the TDL. We intend to perform an empirically controlled experiment to validate

the proposed DSL in a software project of our partnership associated with Systems Engineering Research Center (*Centro de Pesquisa em Engenharia de Sistemas* – CePES[1]), being conducted and supervised by Experimental Software Engineering Lab (*Laboratório de EXperimentos em Engenharia de Software - LEXES*), involving both inexperienced and experienced performance engineers. In a positive scenario, our intention is deploying the approach integrating to performance testing tools derived from Software Product Line (SPL), PLeTs [RVZG10], as a component. The PLeTs is an SPL of model-based testing tools, with an architecture based on components to generate and execute the test scripts and scenarios of performance testing.

The proposed DSL aims to model the behavior of Web application and to derive test scenarios and scripts of performance testing based on models developed. Before of any proposal of creating a domain-specific language, a set of activities was proposed to the research project of CePES, like performance testing for a software project and Proof of Concept (PoC).

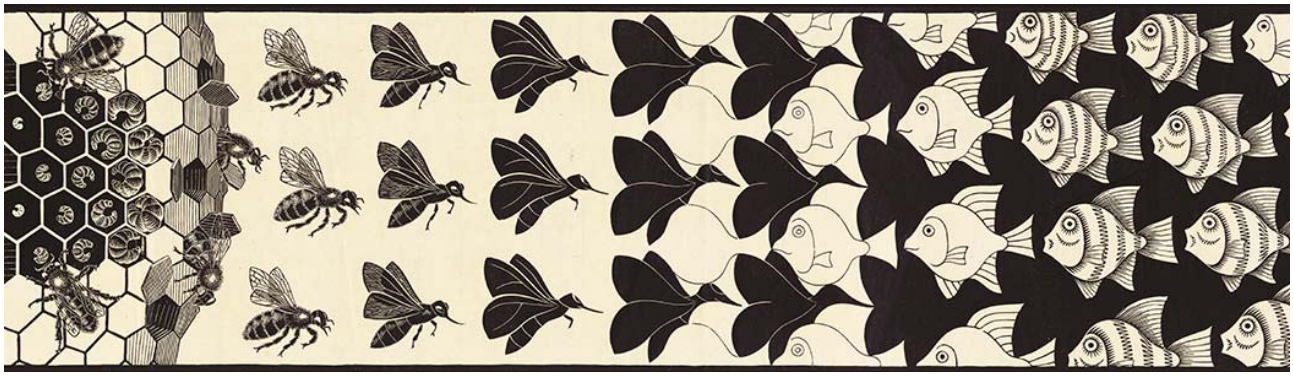
3.4 Chapter Summary

Chapter 3 presented research methodology, research design and research context where this study was developed. Hence, we focused our efforts on formalizing our research, showing how research methods and techniques were applied to plan, conduct and evaluate our research. Thereunto, we divided this chapter into two sections as follows: in the former, we described our research design, presenting how each method was applied in the context of the research evolution; in the latter, we introduced the research setting, contextualizing the environment of the research group as well as our industrial partner.

Chapter 4 discusses the analysis of the performance testing domain and presents the language and their metamodels that compose our DSL, as well as their requirements and design decisions. We also show an example of how to demonstrate the use of our DSL to support the performance testing modeling design of a Web application.

[1] http://www.cepes.pucrs.br

4. CANOPUS



"Canopus (kɛnopu; α Carinae, Alpha Carinae), is a white binary star in the Keel of the ship Argo Navis, is the brightest star in the southern constellation of Carina, and the second brightest star in the night-time sky, after Sirius. Canopus is essentially white when seen with the naked eye. It is located in the far southern sky, at a declination of −52° 42' and a right ascension of 06h24m. Its name comes from the mythological Canopus, who was a navigator for Menelaus, king of Sparta. The Greeks called it Kanobos, and Kanopus, transcribed into Canobus and now it universally is Canopus."
— Wikipedia

4.1 Overview

In Chapter 3, we focused our efforts on formalizing our research, showing how research methods and techniques were applied to plan, conduct and evaluate our DSL. In this chapter, Section 4.2 discusses the analysis of the performance testing domain, as well as its requirements and design decisions in, Section 4.3 and Section 4.4, respectively. Section 4.5 presents Canopus. Next, Section 4.6 presents the metamodels that compose our DSL. Section 4.7 shows an example of use to demonstrate the utilization of our DSL to support the design of performance testing modeling of Web applications. Section 4.8 points out lessons learned. Finally, we summary the chapter in Section 4.9.

4.2 Domain Analysis

Before we start to describe our DSL, it is important to mention some of the steps that were taken prior to the definition of the requirements and design decisions, as well as to clearly define

our problem domain. These steps were taken in collaboration with researchers from our group and test engineers from the Technology Development Lab (TDL) of Dell Computer Brazil, a global IT company.

The aim of the proposed DSL is to allow a performance engineer to model the behavior of Web applications - although, we focus on Web Application, our DSL is not intended to be limited to this domain - and their environment. Through the developed models it is possible to generate test scenarios and scripts that will be used to execute performance testing.

The first step was related to the expertise that was acquired during the development of a Software Product Line (SPL) [CNN01] to derive MBT tools called PLeTs [RVZG10]. These SPL artifacts can be split into two main parts: one that analyzes models and generates abstract test cases from those models, and one that takes the abstract test cases and derive concrete test scripts to be executed by performance testing tools. Several models were studied during the first phase of the development of our SPL, *e.g.* User Community Modeling Language (UCML) [Bar15], UML profiles, such as Schedulability, Performance and Time (SPT) [OMG05] and Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [OMG09], Customer Behavior Model Graphs (CBMG) [MAFM99], and Finite State Machines (FSM) [Gil62]. Likewise, several performance testing environments and tools were studied, such as Apache Jmeter [JLH+10], HP LoadRunner [Hew15], IBM Rational Performance Tester [CDD+08], Borland Silk Performer [BCM07], Microsoft Visual Studio [Mic15b], LoadUI [Bea15], Grinder [GZ15], and Neo Load [Neo15].

The second step was to apply some of the models and tools identified in the previous step to test some open-source applications, such as TPC-W (Transaction Processing Performance-Web) [Men02] and Moodle Learning Management System [Moo15]. Furthermore, we also applied some of the MBT tools generated from our SPL to test those applications and to support the test of real applications in the context of our collaboration with the TDL (see, for example, [RBC+15]). Those real applications were hosted in highly complex environments, which provided us with a deep understanding on the needs of the performance testing teams.

Besides that, we also based our DSL on well-known concepts from Software Engineering Body of Knowledge (SWEBOK) [SBF14], IEEE Standard Glossary of Software Engineering Terminology (IEEE Std. 610.12-1999) [IEE90], IEEE Standard for Software and System Test Documentation (IEEE Std. 829-2008 [IEE08] and other literature, such as Meier *et al.* [MFB+07], Menascè *et al.* [MV00] and Molineux [Mol09]. These references were chosen to mitigate the bias, provide a theoretical basis and ensure the coherency among concepts, features, and properties of the performance domain.

Regarding domain analysis, some works present an approach based on the formalization of domain analysis through ontologies [TMG09] [WSPS09]. Therefore, to determine the concepts, entities and functionalities that represent the performance testing domain, we adopted a strategy to identify and analyze the domain using an ontology [FV14]. This ontology provides the basis for determining the concepts, relationships, and constraints that represent the performance testing domain. Figure 4.1 shows a UML class diagram to resume the main terms represented in the

performance testing ontology [FV14]. Each class in the diagram represents a concept and each association represents a property. Furthermore, the origin of a property denotes its domain in OWL (Ontology Web Language) [BvHH+04], while its destination is the image (or range) of such relationship between concepts.

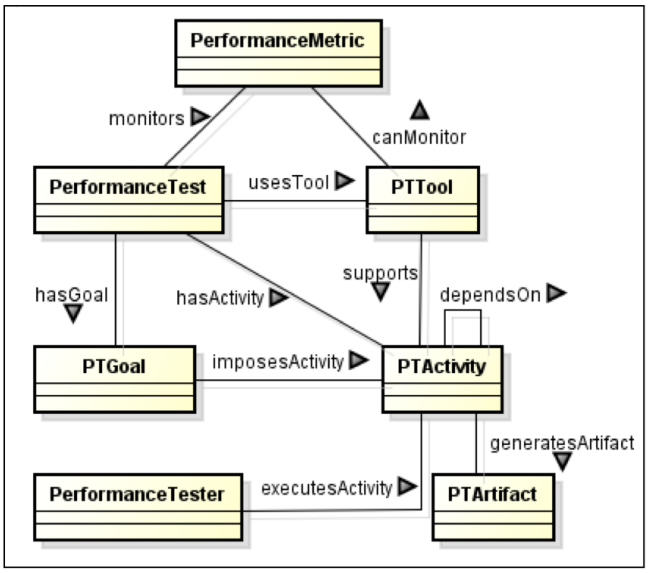


Figure 4.1: Class diagram of main ontology concepts and relationships [FV14]

In addition to this ontology, we used an initial performance testing body of knowledge based on steps aforementioned to provide the basis for determining the objects, relationships, and constraints to express the performance testing domain. Furthermore, this body of knowledge was used as a base to define the performance testing requirements and design decisions for our DSL [BZR+14]. Before creating any DSL to support modeling performance testing in the target DSL, the TDL first considered the use of off-the-shelf solutions, provided that specific requirements were met. Thus, we describe these requirements in detail in the next sections.

4.3 Language Requirements

This section enumerates the requirements (RE) we collected from our expertise and also from the software engineers from TDL. These requirements are related to features and concepts from performance testing domain. Moreover, we discuss some mechanisms for implementing the proposed DSL.

RE1) *The DSL must allow to represent the performance testing features.* One of the main functions of the performance testing is to reveal bottlenecks of a system. Therefore, the applications should be measured and controlled in small parts that can be defined as transactions. This allows to measure the performance quality for each activity of a system. For instance, to define the response time SLA based on these transactions.

RE2) *The technique for developing our DSL must be based on Language Workbench (LW).* Since we do not want to develop new tools, *i.e.*, editor or compiler, as in an external DSL; neither we intend to embed our DSL in a GPL, we will base our DSL on a LW. This will allow us to focus on the analysis domain and development of the new DSL rather than spend effort on implementing new tools or having to choose a General-Purpose Language (GPL) that might not be appropriate for the DSL that we want.

RE3) *The DSL must support a graphical representation of the performance testing features.* This requirement does not concern the language itself, but the LW that will support its development. Thereunto, we desire that the LW supports a graphical-based editor for creating DSL. Moreover, the LW should allow to implement the domain concepts, their translation rules, designing symbols and elements of the language, and also to generate different code for different tools.

RE4) *The DSL must support a textual representation.* The proposed DSL should also include a custom language that is close to a natural language. This will facilitate its adoption by test engineers that are used to use textual representation. The language should have features and keywords that remember the performance testing domain.

RE5) *The DSL must include features that illustrate performance counters.* In performance testing there are many performance counters, *e.g.*, response time or network throughput, that provide means to analyze both application quality level and host infrastructure.

RE6) *The DSL must allow the modeling of the behavior of different user profiles.* This requirement is a specific function of the performance domain, which should allow that the behavior of different user profiles, such as a buyer or new clients, is modeled according to the SUT. In our context we will focus on Web applications.

RE7) *Traceability links between graphical and textual representations should require minimal human intervention/effort.* Traceability is an important feature in software solutions, mainly when involve model transformation, *e.g.*, translation from a graphical to a textual representation. The proposed DSL should automate the mapping process of graphical elements of the model to their respective textual counterparts.

RE8) *The DSL must be able to export models to formats of specific technologies.* This requirement should ensure that models written in our proposed DSL can be exported to the format of the input of specific testing tools, *e.g.*, HP LoadRunner [Hew15], Microsoft Visual Studio [Mic15b], or Neo Load [Neo15].

RE9) *The DSL must generate model information in a XML file.* This requirement aims to ensure that we can export our DSL to any other technology in the future. That is, we export all information from the system model into an *eXtensible Markup Language* (XML) file, so anyone that wants to use our solution can import the XML into their technology.

RE10) *The DSL must represent different performance test elements in test scripts.* The modeled diagram using the proposed DSL must represent multiples elements of test scripts, such as conditional or repetition control flows, among others.

RE11) *The DSL must allow the modeling of multiple performance test scenarios.* Performance testing is responsible to carry out testing of part of or the whole system under normal and/or stress workload. The DSL, therefore, should be able to generate multiples performance test scenarios, *i.e.*, under normal and stress workload conditions.

Currently, to the best of our knowledge, no existing language or model (commercial or not) meets all of the presented requirements. Therefore, given the automation needs of performance testing, we propose a DSL for modeling performance testing of Web applications.

4.4 Design Decisions

In this section, we describe our design decisions for creating a DSL that supports the requirements discussed in Section 4.3. For each design decisions, we mention the associated requirements that are being dealt with.

DD1) *To use a LW that supports graphical DSL* (RE2, RE3). To attend these requirements we performed a literature review on existing LW, including academic, commercial or open-source. The study of Erdweg *et al.* [ESV+13] presents the state-of-the-art in LW and defines some criteria (the authors call them features) that help someone to decide which tool should be adopted. Given the requirements of our proposed DSL, we chose the MetaEdit+ from MetaCase [KLR96], because it supports most of the proposed features in the study of Erdweg.

DD2) *The features of the performance testing domain will be used in an incremental way* (RE1, RE5). Developing a DSL requires a series of phases, such as analysis, design, implementation, and use [MHS05]. Usually researchers focus their attention to the implementation phase, but only a few of them focus on the analysis of the domain and design of the DSL. Nevertheless, there are some methodologies for domain analysis, which helps to unravel the knowledge about the problem domain analyzed. Among them we can highlight Domain Specific Software Architectures (DSSA) [TTC95], Feature-Oriented Domain Analysis (FODA) [KCH+90], and Organization Domain Modeling (ODM) [SA98]. Some works present an approach based on the formalization of domain analysis through ontologies [TMG09] [WSPS09]. Thus, in order to determine the features that represent the performance testing domain, we adopted a strategy to identify and analyze the domain using an ontology [FV14]. This ontology provides the basis for determining the concepts, relationships, and constraints that represent the performance testing domain. Besides the ontology, we used the performance testing body of knowledge (Section 4.2).

DD3) *To provide a graphical language capable of representing the behavior of user profiles for different performance test scenarios* (RE6, RE11). To attend these requirements we analyzed different models and graphical representations that support performance testing. Among the approaches and techniques, the most relevant for our work were UML profiles. Besides that, it is also important to mention a theoretical language proposed by Scott Barber for modeling users behavior, called UCML. Based on these different approaches and techniques, the graphical language will have visual

elements capable of representing the behavior of different user profiles. Besides the flow of activities that the user performs in the SUT, the graphical language will have visual elements to represent the performance test scenarios settings, including information about the performance testing domain, such as number of virtual users (VU), test duration, metrics to be evaluated (response time, memory available, processor time, among others). It is also possible to include the randomization and percentages of execution for each interaction that a VU executes during performance testing. Another very important feature is that the DSL can represent abstract data that will be instantiated in activity of the performance testing process, for example, during the generation of the performance test scripts.

DD4) *To create a textual representation in a semi-natural language* (RE4). Behavior-Driven Development (BDD) [WH12] is an agile software development process, in which acceptance testing, mainly functional testing, is essential to advance to next phase of a software project, since it facilitates the understanding among testing and development teams and stakeholders. Usually, tests are described in natural language in order to ensure this common understanding regarding the system requirements for all project members. Even though it is common to find languages that use natural language to describe functional testing, *e.g.*, Gherkin [WH12], to the best of our knowledge none of them includes performance testing features. Therefore, we intend to extend this language, to include the performance testing features. Gherkin is interpreted by a command line tool called Cucumber, which automates the acceptance testing execution.

DD5) *To provide automated traceability between the graphical and textual representations* (RE7, RE10). Traceability is an important feature that should be mapped in the implementation of a DSL. Thus, it is required that the LW allows the creation of translation rules among models. In this case, the mapping among the graphical elements with their respective assets of the textual representation must be provided. It is important that this mapping is not an one-to-one mapping. Some graphical elements can be mapped to several instances of the textual elements. For example, a graphical decision point can be mapped to several textual scripts, one for each branch present in the graphical representation. In order to solve this mapping, algorithms such as the Chinese Postman Problem [Edm73] can be used.

DD6) *To support the integration of the DSL with other technologies* (RE8, RE9). It should be able to export the models (test scenarios, abstract test cases, etc.) described in the DSL to other formats, such as XML or HP LoadRunner [Hew15] and Microsoft Visual Studio [Mic15b] input formats. The ability to export data in XML format will allow future users of the language to use it with other technologies or IDEs.

4.5 The Language

> *"Works of imagination should be written in very plain language; the more purely imaginative they are the more necessary it is to be plain."*
>
> — Samuel Taylor Coleridge

This section presents the DSL that we developed to meet the requirements described in Section 4.3 and based on the design decision from Section 4.4. Our DSL is composed of three 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.

This set of elements composes the metamodels of Canopus, which will be described in the next section.

4.6 Metamodels

To define the Canopus, we need a model to create our performance testing models, *i.e.* a metamodel, which can be used to create abstract concepts for a certain domain [KT07]. A metamodel is composed of metatypes, which are used to design a specific DSL. The development process of generating such metamodels is called metamodeling, which is a framework defined by metametamodels to generate metamodels [KBJV06]. There are several metamodeling environments, *a.k.a.* Language Workbenches (LW), such as Generic Modeling Environment (GME) [GME15], Eclipse Modeling Framework (EMF) [EMF15] and MetaEdit+ Workbench [Met15]. To support the creation of our DSL, 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) [KT07] metatypes.

A Graph metatype is a collection of objects, relationships and roles. These are bound together to show which objects a relationship element connects through which roles. A graph is also able to maintain information about which graphs its elements decompose into. A graph is one particular model, usually shown as a diagram. The Object metatype is the main element that can be placed in graphs. Examples of objects are the concepts of a domain that must be represented in a graph. It is worthwhile to highlight that all instances of a created object can be reused in other graphs. The Relationship metatype is an explicit connection among two or more objects. Relationships can be attached to an object via roles. The Role metatype specifies how an object participates in a relationship. A port is an optional specification of a specific part of an object to which a role can connect. Normally, roles connect directly to objects, and the semantics of the connection are provided by the role type. If you want a given role type to be able to connect to different places on an object with different semantics, you can add ports to the object's symbol. For example, an Amplifier object might have a port for analog input, a port for digital input, and an analog output port. Roles connecting to each of these will have different semantics. Ports are defined for an object type, and all instances share those same ports. In Figure 4.2, some examples of the basic elements of these metatypes are labeled.

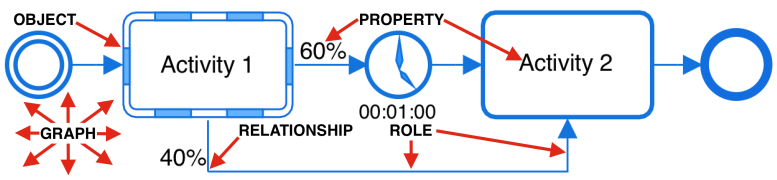


Figure 4.2: GOPPRR metatypes from MetaEdit+ language workbench

Canopus has 7 metamodels presented by 7 packages (see Figure 4.3). The main metamodels that compose our DSL are `Canopus Performance Monitoring`, `Canopus Performance Scenario`, and `Canopus Performance Scripting`, which together compose the `Canopus Performance Model`.

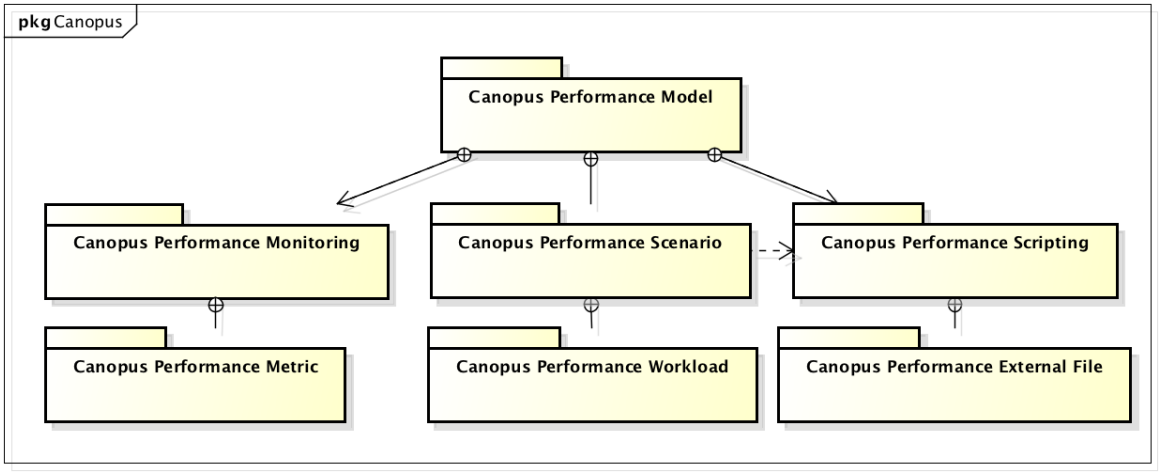


Figure 4.3: Canopus package diagram

Canopus Performance Monitoring Metamodel

The `Canopus Performance Monitoring (CPM)` metamodel is intended to be used to represent the servers deployed in the performance testing environment, *i.e.* application, databases, or even the load generators. Moreover, for each one of these servers, information about the testing environment must be provided, *e.g.* server IP address or host name. It is worth mentioning that even the load generator must be described in our DSL, since in several cases it can be desirable to monitor the performance of the load generator. Furthermore, some load generator solution provides support to generate load from several servers if we really want to stress the application or database server.

The CPM metamodel requires that at least two servers have to be modeled: one that hosts the SUT and another that hosts the load generator and the monitoring tool. The second row of Table 4.1 presents the metatypes supported by the CPM metamodel: 3 objects (`SUT`, `Load Generator (LG)`, `Monitor`); 2 relationships (`Flow`, `Association`); and, 4 roles (`From`, `To`, `Source`, `Target`). Moreover, these metatypes are bound by 4 bindings. For instance, a `Flow` relationship connects, using the `From` and `To` roles, the `LG` to the `SUT` objects. Furthermore, two objects (`LG` and `SUT`) from the metamodel can be decomposed (*i.e*, into subgraphs) in a `Canopus Performance Metric` model.

Canopus Performance Scenario Metamodel

The `Canopus Performance Scenario (CPSce)` metamodel is used to represent the users workload profiles. In the CPSce metamodel, each user profile is associated to one or more scripts. If a user profile is associated with more than one test script, a percentage must be attributed

Table 4.1: Summary of the Canopus metamodels and their GOPPRR metatypes

Metamodel	Objects	Relationships	Roles	Bindings	Subgraphs
Canopus Performance Model	Monitoring Scenario Scripting Group Model Project Label				Monitoring ->Canopus Performance Monitoring Scenario ->Canopus Performance Scenario Scripting ->Canopus Performance Scripting
Canopus Performance Monitoring	SUT Load Generator Monitor	Flow Association	From To Source Target	Association (Source[SUT]->Target[SUT]) Flow (From[LG]->To[SUT]) Flow (From[Monitor]->To[SUT]) Flow (From[Monitor]->To[LG])	LG ->Canopus Performance Metric model SUT ->Canopus Performance Metric model
Canopus Performance Scenario	User Profile Script Workload	Association	From To	Association (From[User Profile]->To[Script])	Script ->Canopus Performance Scripting Workload ->Canopus Performance Workload
Canopus Performance Scripting	Initial Final Activity DataTable Thinktime SaveParameters	Association Fork Join Transition	From To Read Write	Association (Read[Activity]->Write[SaveParameters]) Association (Read[DataTable]->Write[Activity]) Association (Read[SaveParameters]->Write[Activity]) Fork (From[Activity]->To[Activity]) Join (From[Activity]->To[Activity]) Transition (From[Activity]->To[Activity]) Transition (From[Activity]->To[Final]) Transition (From[Activity]->To[ThinkTime]) Transition (From[Initial]->To[Activity]) Transition (From[ThinkTime]->To[Activity])	Activity ->Canopus Performance Scripting DataTable->Canopus External File
Canopus Performance Metric	Metric Counter Threshold Criteria	Association Bifurcation	From Metric To Counter From Counter To Criteria To Threshold	"Association (From Metric[Metric]->To Counter[Counter])" "Bifurcation (From Counter[Counter]->To Criteria[Criteria] && From Counter[Counter]->To Threshold[Threshold])"	
Canopus Performance Workload	Ramp Down Time Ramp Down User Ramp Up Time Ramp Up User Test Duration Virtual User				
Canopus External File	Rows Columns	Traceability	From To	Traceability(From[Rows]->To[Columns])	

to every script belonging to this profile, *i.e.* it defines the number of users that will execute a test script. In addition to setting user profiles, in the CPSce metamodel it is also important to set one or more workload profiles. Each workload profile is decomposed into a subgraph, a `Canopus Performance Workload` (CPW) metamodel, which in turn is composed by 6 objects, defined as follows:

- Virtual Users (VU): refers to the number of VU who will make requests to the SUT;
- Ramp Up Time (RUT): defines the time each set of ramp up users takes to access the SUT;
- Ramp Up Users (RUU): defines the number of VU who will access the SUT during each ramp up time interval;
- Test Duration (TD): refers to the total performance test execution time for a given workload. It is important to mention that the test duration will take, at least, the ramp up time multiplied by the number of intervals of ramp up time plus the ramp down time multiplied by the number of intervals of ramp down time, *i.e.* $TD \geq n \times RUT + m \times RDT$, where $n = \lceil VU/RUU \rceil - 1$ and $m = \lceil VU/RDU \rceil - 1$;
- Ramp Down Users (RDU): defines the number of VU who will leave the SUT on each ramp down time;
- Ramp Down Time (RDT): defines the time a given set of ramp down users takes to leave the SUT.

As shown in Table 4.1, the CPSce metamodel is composed by 3 objects (`User Profile`, `Script`, and `Workload`); 1 relationship (`Association`); 2 roles (`From` and `To`). Besides, there is a unique binding, which connects a `User Profile` to `Script` objects, respectively, through `From` and `To` roles. Moreover, each `Script` object attached to a scenario model must be decomposed into a subgraph, *i.e.* a `Canopus Performance Scripting` metamodel.

Canopus Performance Scripting Metamodel

The `Canopus Performance Scripting` (CPScr) metamodel represents each of the scripts from the user profiles in the scenarios part. The CPScr metamodel is responsible for determining the behavior of the interaction between VU and SUT. Each script includes activities, such as, transaction control or think time between activities. Similarly to the percentage for executing a script, which is defined in the CPSce metamodel, each script can also contain branches that will have a user distribution associated to each path to be executed, *i.e.* the number of VU that will execute each path. During the description of each script, it is also possible to define a set of parallel or concurrent activities. This feature is represented by a pair of fork and join objects. Our DSL also allows an activity to be decomposed into another CPScr metamodel. The CPScr metamodel also supports that a parameter generated in runtime can be saved to be used in other activities of the same script flow (the `SaveParameters` object handles this feature). Table 4.1, fourth row, shows the composition of the CPScr metamodel: 6 objects, 4 relationships, 4 roles, 9 bindings and 2 metamodels.

4.7 Example of Use: TPC-W

> *"Example is not the main thing in influencing others. It is the only thing."*
>
> — Albert Schweizer

In order to present an example of use of the proposed DSL, we selected an e-commerce Web application provided by the TPC-W benchmark [Men02]. TPC-W is a transactional benchmark for e-commerce sites developed to conduct performance testing of the infrastructure in which the Web site will be hosted. It can be used, for instance, to check the number of users that an environment can attend, or if it has the amount of resources required for providing that service. TPC-W simulates a set of user activities, reproducing the main operations performed during a site visit.

This section presents a small sample of both the graphical and textual representations described in previous sections. We instantiate the modeling performance testing process through the proposed DSL for the TPC-W e-commerce Web application. The goal is to give some understanding of the requirements and design decisions of a language to model user behavior, as well as, the performance test scenarios. The graphical representation contains elements for virtual users, test duration, ramp up time, ramp up users, ramp down time, think time, among others.

4.7.1 Canopus Performance Monitoring Model

Figure 4.4 shows the `Canopus Performance Monitoring` model, which illustrates the performance testing environment where the TPC-W application is deployed to. This environment is composed of seven hosts: `ZION` and `BACO` are a Cloud Service (CS) hosting the MySQL TPC-W's database; `ZEUS` and `POSEIDON` are a Virtual Machine (VM) hosting a Web Application server (Apache service); `BIG_BROTHER` is a Physical Machine (PM) server hosting a performance monitoring tool; and, similarly, the `RUNNER` and `LOAD` are physical servers hosting a workload generator tool. The `RUNNER` and `LOAD` load generatores are used to generate VU (synthetic workload) to exercise the `ZEUS` and `POSEIDON` Web servers, and their monitoring module is hosted on the `BIG_BROTHER` server for monitoring the level of resources used by `ZEUS`, `POSEIDON`, as well as `ZION` and `BACO` database servers.

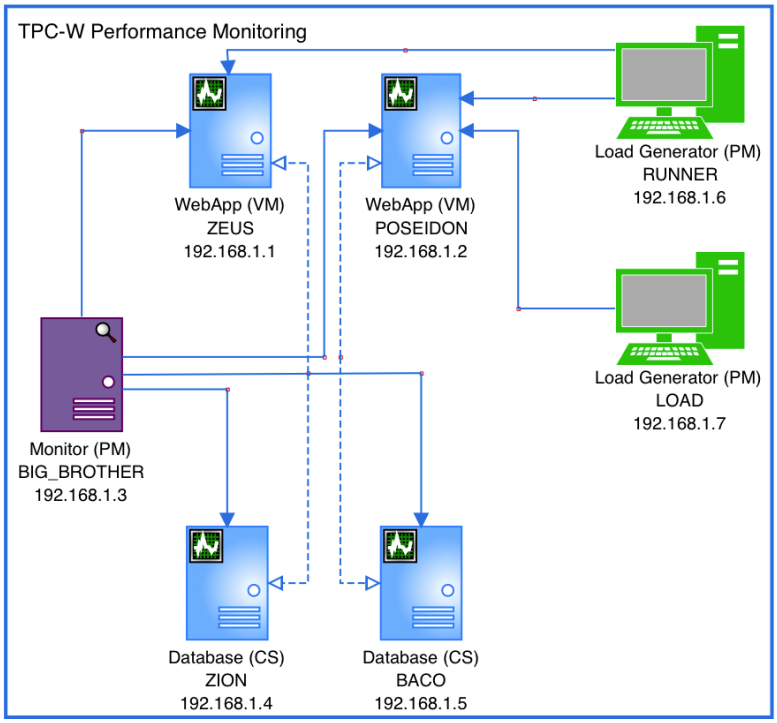


Figure 4.4: Graphical representation of the TPC-W `Canopus Performance Monitoring` model

The monitoring module can be set up with a set of performance metrics, as well as the acceptable thresholds of computational resources, *e.g.* processor, memory, and disk usage. For instance, Figure 4.5 presents a single metric of the `Canopus Performance Metric` model. This figure depicted four types of objects; metric, counter, criterion and threshold. In this case, the metric is `memory` that is related to the `Available MBytes` counter. In turn, each counter can be related to two or more criteria and threshold. This model presents a memory metric composed of three criteria based on available memory, each one of them is associated to a number of VU that are accessing the SUT. For instance, when there are less than 350 users, a host must have at least 3000 MB of available memory. A snippet of a textual representation of the `Canopus Performance Monitoring` model is depicted in Figure 4.6. It is important to highlight that the textual representation supports the

definition of several metrics, including the set up infrastructure information (hostname, IP address) based on the monitoring model (Figure 4.4) such as monitors, load generators, and SUT servers. The range of lines between seven and ten from Figure 4.6 represents information annotated based on the metric model from Figure 4.5.

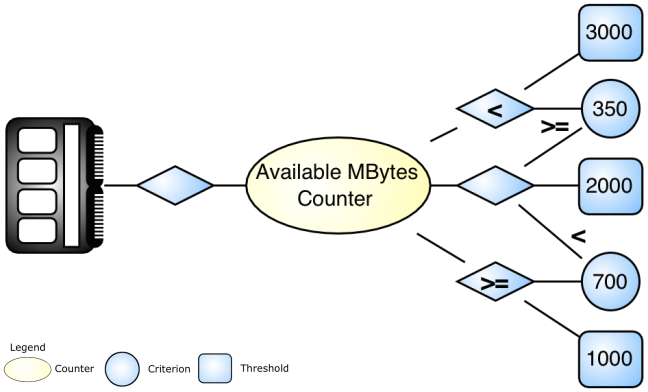


Figure 4.5: Graphical representation of a `Canopus Performance Metric` model of the memory available Mbytes counter

```
1    Feature: Monitor the performance counters of the system and environment.

2  Monitoring: Control the performance counters of the application.
3       Given that "ZEUS:192.168.1.1" WebApp monitored by "BIG_BROTHER:192.168.1.3" monitor
4         And workload generated through "RUNNER:192.168.1.6" load generator for the WebApp on
       "ZEUS"
5         And workload generated through "LOAD:192.168.1.7" load generator for the WebApp on
       "ZEUS"
6         And the #TPC-W Canopus Performance Scenario# test scenario
7        When the "Memory" is monitored
8        Then the "Available MBytes Counter" should be at least "2000" when the number of
       virtual users are between "350" and "700"
9         And at least "1000" MB when the number of virtual users is greater than or equal to
        "700"
10        And at least "3000" MB when the number of virtual users is less than "350"
11       When the "Disk" is monitored
12       Then the "% Idle Time Counter" should be less than "90" when the number of virtual
       users is less than "1000"
13       When the "Web Resources" is monitored
14       Then the "Throughput Mbytes" should be less than "60000" when the number of virtual
       users is less than "1000"
15       When the "Transaction" is monitored
16       Then the "Transaction Per Second (TPS)" should be less than "55000" when the number of
        virtual users is less than "1000"
17       And the "Transaction Response Time" should be less than "3" when the number of virtual
        users is less than "1000"
18       When the "Processor" is monitored
19       Then the "% Processor Time Counter" should be less than "70" when the number of
       virtual users is less than "1000"
```

Figure 4.6: Snippet of textual representation of the TPC-W `Canopus Performance Monitoring` model

4.7.2 Canopus Performance Scenario Model

Figure 4.7 gives an example of a `Canopus Performance Scenario (CPSce)` model that represents the SUT functionalities divided into three scripts: `Browser`, `Shop` and `Order`. Each script has a percentage of the VU that will execute such script. For each script, a VU has a percentage of buying a product from the site. This percentage is bigger for script `Order` and smaller for script `Browser`. The model also shows the interaction behavior of three different user profiles: `Browsing`, `Shopping` and `Ordering`. In short, the user profiles differs from one another on the percentage that they will order, shop or browse.

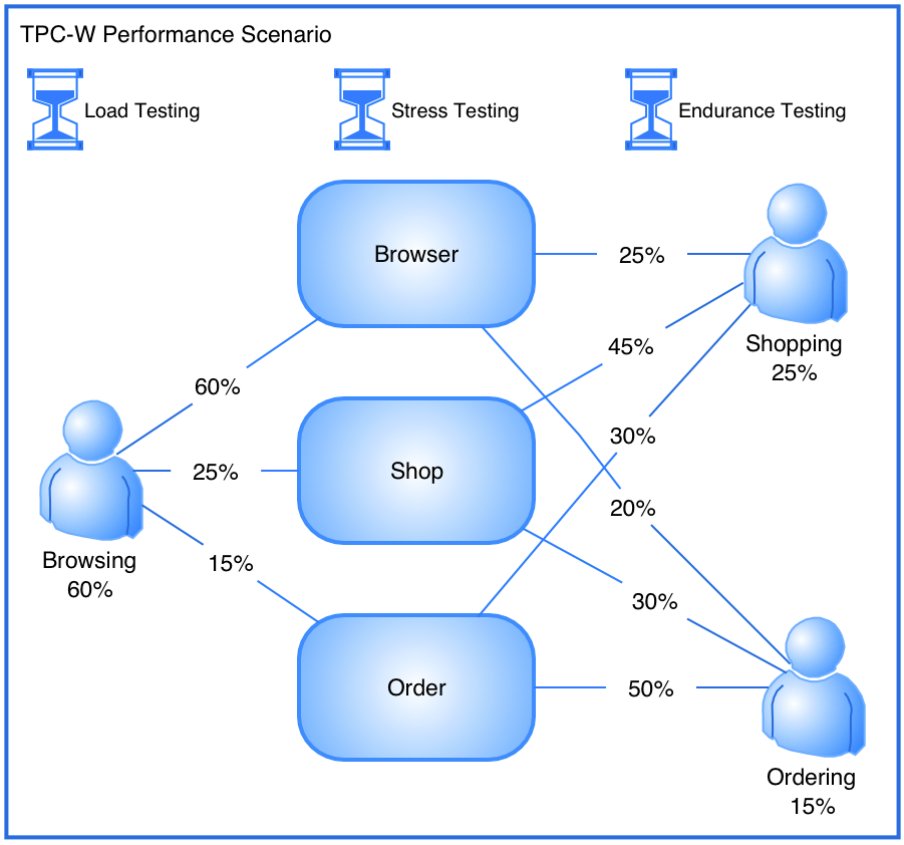


Figure 4.7: Graphical representation of the TPC-W `Canopus Performance Scenario` model

The CPSce is also composed of three workload profiles: `Load Testing`, `Stress Testing`, and `Endurance Testing`. Each workload profile associated to the percentage of each user profile determining the total number of VU. Figure 4.8 shows the `Load Testing` workload profile, a `Canopus Performance Workload` model. This workload model defines that 1000 VU will be running over TPC-W during 4 hours (test duration time). Besides, this model also presents information about the users ramp up and ramp down. For instance, during the test execution 100 VU will be added during every 1-minute interval, while the ramp down defines that 200 VU will leave the SUT during 30-seconds interval. Thus, the `Load Testing` workload defines that 1000 VU will be simultaneously executing during 3h47min30s, the users ramp up time (10min) and the users ramp down

time (2min30s), therefore, completing 4 hours of test duration time. In short, the other workload testing differ from one another on the number of VU and test duration time.

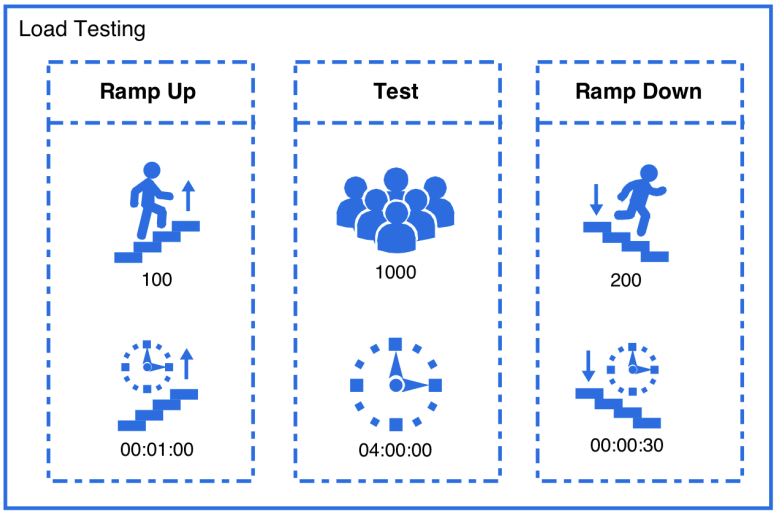


Figure 4.8: Graphical representation of the TPC-W `Canopus Performance Worload` model

Figure 4.9 presents a snippet, in textual representation, for the graphical CPSce model shown in Figure 4.7. The textual representation, as mentioned before, is structured based on the Gherkin language, but with elements that represent the semantics for performance testing, *e.g.*, `RUUsers` or `RUTime`. The name of the files that store the performance test scripts is shown between `#`. Furthermore, Figure 4.9 also shows all information that describe the scenarios settings and distributions scripts in a table format. This is interesting since we can describe different scenarios settings and distribution scripts in just one place. Notice that each column in the table format is associated with a name, delimited by { and }, that are used in the scenario description.

Regarding Figure 4.10 that depicts the whole workload profiles associated to the same performance scenario. The great advantage of this textual representation is that the number are not just a variable (parameter), but a solved number dynamically generated.

4.7.3 Canopus Performance Scripting Model

Figure 4.11 presents a snippet of the `Shop` script. The model is composed of six activities, five of them with transaction control, shown by dashed border figures. The model also contains a `Think Time` of 5 seconds (Clock element) and three `Data Table` elements, *e.g.*, `Books by Category.CSV`. In some cases, there is the necessity to store the results of processing of an activity into global variables or parameters, so that other activities can use these data. For example, to decide a path to a decision point (see `Category Choice`), this feature is called `Save Parameters` that is represented by a floppy disk element. In such script, there is an activity with loop feature drawn on the bottom activity element, which allows to perform many instances of the same activity.

A textual representation of the `Canopus Performance Scripting` model from Figure 4.11 is shown in Figure 4.12. An advantage of using a textual representation is to present all perfor-

```
1  Feature: Execute the scenario test for a workload.

2 Scenario: Evaluate the {Scenarios.Workload} workload for {Scenarios.VU} users simultaneously
3    Given {Scenarios.RUUsers} users enter the system every {Scenarios.RUTime}
4      And {Scenarios.RDUsers} users leave the system every {Scenarios.RDTime}
5      And {Scenarios.VU} users register into the system simultaneously
6      And performance testing execute during {Scenarios.Time}
7     When each user executes a script likelihood of profile {Distribution.Profile}
8     When {Distribution.Percentage} of the virtual users execute the {Distribution.Profile}
       user profile:
9     Then {Distribution.Browser} of them execute the #Browser# script
10     And {Distribution.Shop} of them execute the #Shop# script
11     And {Distribution.Order} of them execute the #Order# script

12 Examples: Scenarios.csv
13 | Workload          |      VU | Time     | RUUsers | RUTime   | RDUsers |   RDTime |
14 | Load Testing      |    1000 | 04:00:00 |     100 | 00:01:00 |     200 | 00:00:30 |
15 | Stress Testing    |    1000 | 02:00:00 |     100 | 00:00:15 |     200 | 00:00:15 |
16 | Endurance Testing |    8000 | 20:00:00 |     200 | 00:01:00 |     200 | 00:00:30 |

17 Examples: Distribution.csv
18 | Profile   | Percentage | Browser | Shop | Order |
19 | Browsing  |        60% |     60% |  25% |   15% |
20 | Shopping  |        25% |     25% |  45% |   30% |
21 | Ordering  |        15% |     20% |  30% |   50% |
```

Figure 4.9: Textual representation of a parameterized Canopus Performance Scenario model

mance testing information annotated in the model, facilitating the view of all relevant information. Although the graphical representation provides a better way to represent the main domain concepts, there is some information that are better represented by elements from the textual representation. Further, some activities have a dynamic parameter that refers to a data table, such as the New Products activity (line 5) that refers to the {Books by Category.Book_Id} parameter (line 6), which in turn refers to a data table presented at the end of the script (line 30). Note that the test data associated to the {Books by Category.Book_Id} parameter will be updated on each interaction of a VU based on a random strategy. Last but not least, the script solves the loop anotation tagged into Search Request activity, it is represented by loop symbol on the bottom activity element as can be seen in Figure 4.11. Hence, the interval lines from 10 to 15 (Figure 4.12) represents this loop feature. If someone wants to repeat not only one button a set of activities, it is possible to associate a subgraph model into an activity. This feature is represented by "+" (plus symbol) on the bottom Search Request activity.

4.8 Lessons Learned

In short, the main lessons we have learned from the development and the use of Canopus are as follows:

```
1   Feature: Execute the test scenario for a workload.

2  Scenario: Evaluate the #Load Testing# workload for "1000" users simultaneously
3     Given "100" users enter the system every "00:01:00"     /* Ramp Up hh:mm:ss */
4       And "200" users leave the system every "00:00:30"      /* Ramp Down hh:mm:ss */
5       And "1000" users register into the system simultaneously  /* Virtual Users */
6       And performance testing execution during "04:00:00"       /* Test Duration hh:mm:ss */
7      When "60%" ("600") of the virtual users execute the #Browsing# user profile:
8      Then "60%" ("360") of them execute the #Browser# script
9       And "25%" ("150") of them execute the #Shop# script
10      And "15%" ("90") of them execute the #Order# script
11     When "15%" ("150") of the virtual users execute the #Ordering# user profile:
12     Then "20%" ("30") of them execute the #Browser# script
13      And "30%" ("45") of them execute the #Shop# script
14      And "50%" ("75") of them execute the #Order# script
15     When "25%" ("250") of the virtual users execute the #Shopping# user profile:
16     Then "25%" ("63") of them execute the #Browser# script
17      And "45%" ("113") of them execute the #Shop# script
18      And "30%" ("75") of them execute the #Order# script

19 Scenario: Evaluate the #Stress Testing# workload for "1000" users simultaneously
20    Given "100" users enter the system every "00:00:15"     /* Ramp Up hh:mm:ss */
21      And "200" users leave the system every "00:00:15"      /* Ramp Down hh:mm:ss */
22      And "1000" users register into the system simultaneously  /* Virtual Users */
23      And performance testing execution during "02:00:00"       /* Test Duration hh:mm:ss */
24     When "60%" ("600") of the virtual users execute the #Browsing# user profile:
25     Then "60%" ("360") of them execute the #Browser# script
26      And "25%" ("150") of them execute the #Shop# script
27      And "15%" ("90") of them execute the #Order# script
28     When "15%" ("150") of the virtual users execute the #Ordering# user profile:
29     Then "20%" ("30") of them execute the #Browser# script
30      And "30%" ("45") of them execute the #Shop# script
31      And "50%" ("75") of them execute the #Order# script
32     When "25%" ("250") of the virtual users execute the #Shopping# user profile:
33     Then "25%" ("63") of them execute the #Browser# script
34      And "45%" ("113") of them execute the #Shop# script
35      And "30%" ("75") of them execute the #Order# script

36 Scenario: Evaluate the #Endurance Testing# workload for "8000" users simultaneously
37    Given "200" users enter the system every "00:01:00"     /* Ramp Up hh:mm:ss */
38      And "200" users leave the system every "00:00:30"      /* Ramp Down hh:mm:ss */
39      And "8000" users register into the system simultaneously  /* Virtual Users */
40      And performance testing execution during "20:00:00"       /* Test Duration hh:mm:ss */
41     When "60%" ("4800") of the virtual users execute the #Browsing# user profile:
42     Then "60%" ("2880") of them execute the #Browser# script
43      And "25%" ("1200") of them execute the #Shop# script
44      And "15%" ("720") of them execute the #Order# script
45     When "15%" ("1200") of the virtual users execute the #Ordering# user profile:
46     Then "20%" ("240") of them execute the #Browser# script
47      And "30%" ("360") of them execute the #Shop# script
48      And "50%" ("600") of them execute the #Order# script
49     When "25%" ("2000") of the virtual users execute the #Shopping# user profile:
50     Then "25%" ("500") of them execute the #Browser# script
51      And "45%" ("900") of them execute the #Shop# script
52      And "30%" ("600") of them execute the #Order# script
```

Figure 4.10: Textual representation of the TPC-W Canopus Performance Scenario model

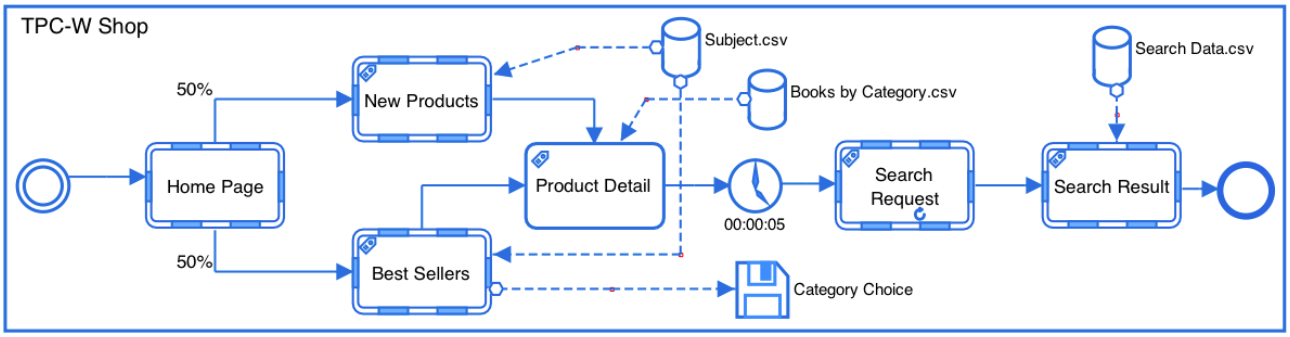


Figure 4.11: Graphical representation of the TPC-W Shop `Canopus Performance Scripting` model

LL1) *Requirements elicitation.* There are several techniques to achieve the best possible results from requirements elicitation process, such as interviews, ethnography, domain analysis, among others. We used domain analysis to understand and capture our domain knowledge, and to identify of reusable concepts and components. Thus, we learned that eliciting requirements based on domain analysis, having as output an initial performance testing body of knowledge, was effective to identify what we needed for our DSL;

LL2) *Domain analysis.* Domain analysis based on ontologies is a good alternative to transform the concepts and relationships from the ontology into entities and functionalities of the DSL. There are several methods and techniques for describing this approach, for instance [TMG09] [WSPS09];

LL3) *Learning curve.* One of the disadvantages of using DSL is the high cost of training users who will use the DSL, *i.e.*, steep learning curve [Gho11]. However, based on our previous experience using several load generator tools in an industrial setting, this disadvantage can be handled pragmatically, since the cost for a new staff to learn several load generators technologies is higher than compared to our DSL. Nonetheless, this drawback must be demonstrated with empirical evidences;

LL4) *Incremental development methodology for creating DSL.* We adopt an incremental development methodology [Bie06] for developing our proposed DSL. This methodology allows us to improve the DSL on each interaction, which is composed by the following steps: analysis, implementation, and use [vDKV00];

LL5) *Requirements and Design Decisions.* Through a pilot study and based on our performance testing body of knowledge, we collected specific needs for performance modeling adoption to create a DSL for modeling performance testing, and argue that existing models and languages do not meet the specificity of the requirements at hand. We then presented our design decisions for creating a DSL. We claim that the reported requirements and design decisions as the two of the contributions of this work, since currently few studies bring such discussion. Our work adds to that in the sense that the elicited requirements can evidence practical scenarios that other load generators may consider supporting; the design decisions, in turn, may be reused or adapted to improve existing DSL, models or languages, or even new ones, targeting similar requirements.

```
Feature: Execute the performance test script for different user profiles

 Script: the users performs purchase interactions based on Test Case N1 from #TPC-W Shop#
  Given the #Home Page# transaction activity through "http://localhost:8080/tpcw/
    TPCW_home_interaction" action, which is loaded in "1.5" seconds
   When I click on "Subject" link {Subject.Category}, which is dynamically generated and
    update on "Each Interaction" based on a "Random" strategy
   Then I will be taken to "/TPCW_new_products_servlet" action in the #New Products#
    transaction activity, which is loaded in "5" seconds
    And I click on "Product" link {Books by Category.Book_Id}, which is dynamically
    generated and update on "Each Interaction" based on a "Random" strategy
    And I need to wait thinktime during "00:00:05"
   Then I will be taken to "/TPCW_product_detail_servlet" action in the #Product Detail#
    transaction activity, which is loaded in "3" seconds
    /* Repeat block three times to perform searching of random books */
    And I press the "Search" button {Search}
   Then I will be taken to "/TPCW_search_request_servlet" action in the #Search Request#
    transaction activity, which is loaded in "2.7" seconds
    And I press the "Search" button {Search}
   Then I will be taken to "/TPCW_search_request_servlet" action in the #Search Request#
    transaction activity, which is loaded in "2.7" seconds
    And I press the "Search" button {Search}
   Then I will be taken to "/TPCW_search_request_servlet" action in the #Search Request#
    transaction activity, which is loaded in "2.7" seconds
    And I select "Type" drop-down list with {Search Data.Type}, which is dynamically
    generated and update on "Each Interaction" based on a "Random" strategy
    And I fill in "Search" input field within {Search Data.Search}, which is dynamically
    generated and update on "Each Interaction" based on a "Same As" "TYPE" strategy
    And I press the "Submit" button {Submit}
   Then I will be taken to "/TPCW_execute_search" action in the #Search Result# transaction
    activity, which is loaded in "2.9" seconds
[50.0%]

 Script: the users performs purchase interactions based on Test Case N2 from #TPC-W Shop#
  Given the #Home Page# transaction activity through "http://localhost:8080/tpcw/
    TPCW_home_interaction" action, which is loaded in "1.5" seconds
 ...
 [50.0%]
Examples: Subject.csv
| CATEGORY            | SUBCATEGORY         |
| Suspense            | Thriller            |
| Romantic            | Historical          |
| Fiction             | Action Adventure    |

Examples: Books by Category.csv
| SUBJECT             | BOOK_ID             |
| COMPUTERS           | 2187                |
| COMPUTERS           | 1145                |
| COMPUTERS           | 2123                |

Examples: Search Data.csv
| TYPE                | SEARCH              |
| Author              | John                |
| Title               | Performance Testing |
| Subject             | Performance         |
```

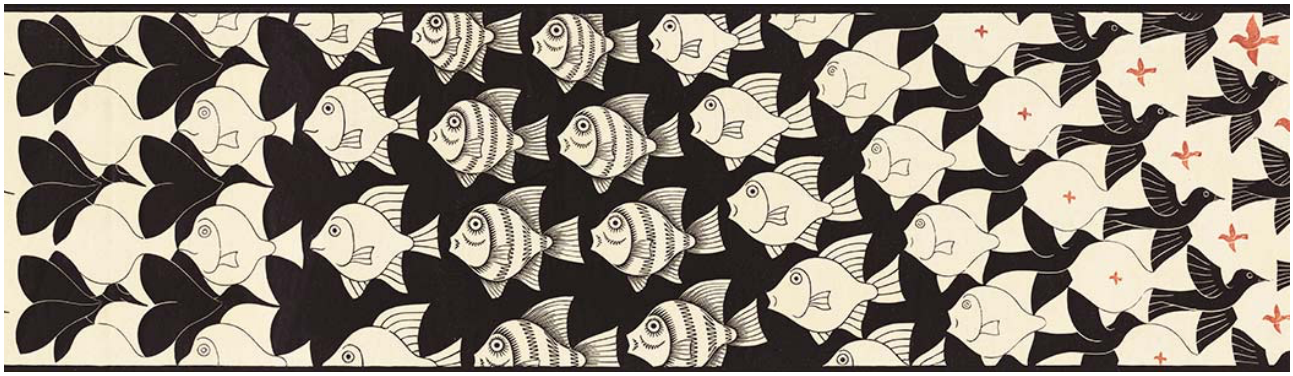
Figure 4.12: Snippet of textual representation of the TPC-W Shop Canopus Performance Scripting model

4.9 Chapter Summary

This chapter discussed the analysis of the performance testing domain and presented a set of elicited requirements in the context of an industrial partner, its design decisions, as well as, the language and its metamodels that compose our DSL. In order to demonstrate that our DSL is suitable to design the performance testing modeling, we selected an e-commerce, our SUT, as an example of use to model the performance testing using Canopus. We also showed and explained how to instantiate for each the Canopus metamodel its respective model of the SUT. Finally, we point out the main lessons learned that we identified during the development and utilization of our DSL.

Chapter 5 presents a case study conducted in collaboration with our partner company. In this case study, we show how to apply our model-based performance testing process that uses Canopus to model performance testing to test an industrial Web application.

5. CASE STUDY



"If you wait until there is another case study in your industry, you will be too late!"

— Seth Godin

5.1 Overview

In this chapter, we present how Canopus was designed to be applied with an MBT approach to model performance test scenarios and scripts. To demonstrate how our DSL can be used in practice, we applied it throughout an actual case study from the industry, in the context of a project of collaboration between a Technology Development Lab (TDL) of Dell Computer Brazil and our university. Inittialy, we describe our model-based performance testing process (Section 5.2) using Canopus. After, this we present an industrial case study (Section 5.3) to evaluate the applicability of Canopus to model performance testing. Section 5.4 presents our case study analysis. Section 5.5 points out lessons learned. Finally, we summarize the chapter in Section 5.6.

5.2 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 5.1 shows our process for modeling performance testing using Canopus. The process incorporates a set of activities that have to be performed by two different parties: `Canopus` and `Third-Party`. The main activities that define our model-based performance testing process are: `Model Performance Monitoring`, `Model Performance Scenario`, `Model Performance Scripting`, `Generate Textual Representation`, `Generate Third-Party Scripts`, `Generate Canopus XML`, `Execute Generated Scripts`, `Monitor Performance Counters`, and `Report Test Results`. The details related to the activities of our DSM process are described next.

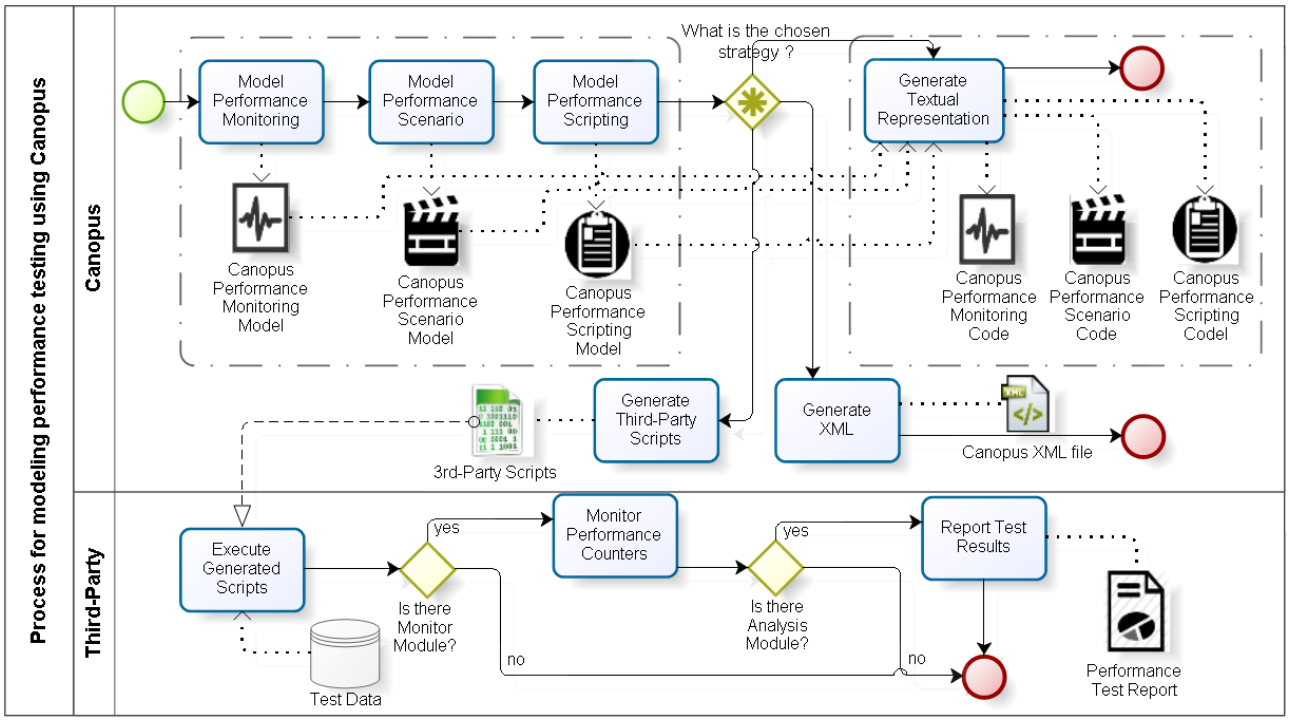


Figure 5.1: Model-based performance testing process using Canopus

5.2.1 Model Performance Monitoring

The designing of the performance monitoring model is the first activity of our process, which 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 `Canopus Monitoring Model`. 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 `Canopus Performance Metric` that can be associated with one or more of these objects. A `Canopus Performance Metric` model represents a set of predefined metrics, for instance, memory, processor, network, throughput, etc. Each one of them is associated with a metric counter, which in turn is linked to a criterion and a threshold.

5.2.2 Model Performance Scenario

The next activity of our process consists of modeling the performance test scenario. The `Canopus Scenario 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 `Canopus Scripting Model`. Besides, each scenario allows modelling several workloads in a same model. A workload (`Canopus`

`Performance Workload`) is constituted of setup objects of test scenario, *i.e.* ramp up, ramp down, test duration (time) and number of VU.

5.2.3 Model Performance Scripting

In this activity, each script object, modeled in the `Canopus Performance Scenario Model`, mimics (step-by-step) the dynamic interaction by VU with the SUT. This activity generates a `Canopus Scripting Model` that is composed of several objects, such as, activity, think time, save parameters and data table. 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 `Canopus Scripting Model` that allows encapsulating a set of activities to propose its reuse into other models. The latter is associated with a `Canopus External File` 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 models (Monitoring, Scenarios and Scripts), 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.

5.2.4 Generate Textual Representation

This activity consists of generating a textual representation in a semi-natural language, a DSL based on the Gherkin [WH12] 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, testing teams and stakeholders. It is important to highlight that a partial textual representation can be generated, for example, to a particular Canopus Performance model.

5.2.5 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 such as HP LoadRunner [Hew15], Microsoft Visual Studio [Mic15b] or Neo Load [Neo15].

5.2.6 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. Hence, Canopus can export entire performance testing information from Canopus Performance models to an XML file. The ability to export data in XML format might allow future Canopus users to use other technologies or IDEs. For instance, in a previous work, we developed a model-based performance testing tool, called PLeTsPerf tool [RBC+15]. This tool can parse our Canopus XML file and process the automatic generation of performance test scenarios and scripts to HP LoadRunner [Hew15].

5.2.7 Third-Party

The other three activities, shown in Figure 5.1, are not part of the Canopus process and are just an example of one possible set of steps that can be executed by a performance engineer, depending on the third-party tool that is used. For example, 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 `Canopus Performance Scripting` 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. Some of these activities can use annotated information in the Canopus Performance models, *e.g.* the `Canopus Performance Monitoring` model and the `Canopus Performance Scripting` model.

5.3 Case Study

In this section we present an industrial case study, using the Changepoint application, in which Canopus is applied to model performance testing information supported by the model-based performance testing presented in Section 5.2.

5.3.1 Changepoint

Changepoint [Cha15] is a commercial solution to support Portfolio and Investment Planning, Project Portfolio Management and Application Portfolio Management. This solution can be adopted as a "out of the box solution" or it can be customized based on the client needs. In the context of our case study, Changepoint is customized in accordance with the specific needs of a large IT company. Moreover, as Changepoint is a broad solution, in our case study we focus on how our DSL is applied to model the Project Portfolio Management module. That is, we show how

to instantiate the modeling performance testing process. The goal of this case study is to evaluate and also demonstrate how our DSL can be used throughout an actual case study in an industrial setting.

To provide a better understanding of how Canopus was applied in the context of our case study, we show how to model performance testing using each one of the Canopus Performance metamodels described in Section 4.6. In short, we intend to explore the following Research Questions (RQ):

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?*

5.3.2 Canopus Performance Monitoring

Figure 5.2 shows the performance testing environment where the Changepoint application is deployed to. This environment is composed of five hosts: *Database_Server* is a physical server hosting the Changepoint database; *Web_Server* is a virtual machine hosting a web server; *APP_Server* is a virtual machine hosting the applications server; *Spy* is a physical server hosting a performance monitoring tool; and, the *Workload* is a physical server hosting a workload generator tool.

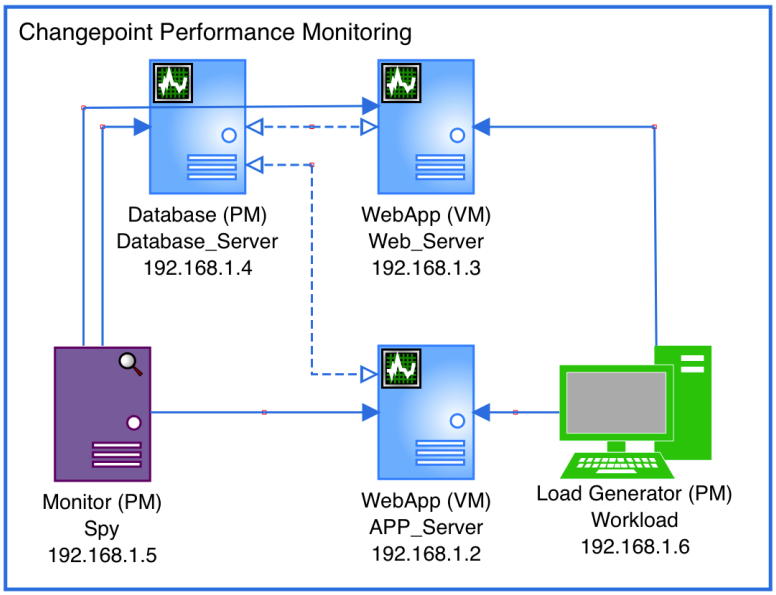


Figure 5.2: Graphical representation of the Changepoint `Canopus Performance Monitoring` model

The LoadRunner load generator (*Workload*) is used to generate VU (synthetic workload) to exercise the *Web_Server* and *APP_Server* servers, and its monitoring module is hosted on the *Spy* server for monitoring the level of resources used by the *Web_Server*, *APP_Server* and *Database_Server* servers. The monitoring module can be set up with a set of performance metrics, as well as the acceptable thresholds of computational resources, *e.g.* processor, memory, and disk

usage. For instance, Figure 5.3 presents a single metric of the Canopus Performance Metric model. This figure depicted four elements; metric, counter, criterion and threshold. In this case, the metric is Processor that is related to the % Processor Time counter. This measurement is the amount of processor utilization, which is set so that the percentage of time the processor is busy must be lower than 70 percent for up to one thousand VU.

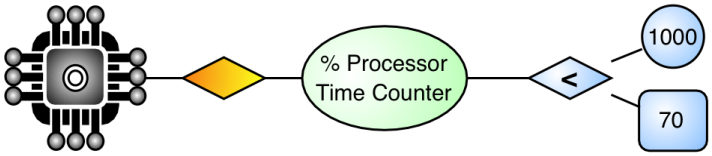


Figure 5.3: Graphical representation of the Canopus Performance Metric model

In turn, each counter can be related to two or more criteria and thresholds. A snippet of a textual representation of the Canopus Performance Monitoring model is depicted in Figure 5.4. For instance, line 18 of this figure represents the translation of the Canopus Performance Metric model shown in Figure 5.3, which describes the percentage of processor time counter for the Processor metric. It is important to highlight that the textual representation supports the definition of several metrics, including the set up infrastructure information (hostname, IP address) based on the monitoring model (Figure 5.2) such as monitors, load generators, and SUT servers.

```
1    Feature: Monitor the performance counters of the system and environment.

2  Monitoring: Control the performance counters of the application.
3       Given that "APP_Server:192.168.1.2" webapp monitored by "Spy:192.168.1.5" monitor
4         And workload generated through "Workload:192.168.1.6" load generator(s) for the
       WebApp on "AppServer"
5         And the #Changepoint Performance Scenario# test scenario
6        When the "Memory" is monitored
7        Then the "Available MBytes Counter" should be at least "2000" MB when the number of
       virtual user are between "350" and "700"
8         And at least "1000" MB when the number of virtual user is greater than or equal to
       "700"
9         And at least "3000" MB when the number of virtual user is less than "350"
10       When the "Disk" is monitored
11       Then the "% Idle Time Counter" should be less than "90" when the number of virtual
       user is less than "1000"
12       When the "Web Resources" is monitored
13       Then the "Throughput Mbytes" should be less than "60000" when the number of virtual
       user is less than "1000"
14       When the "Transaction" is monitored
15       Then the "Transaction Per Second (TPS)" should be less than "55000" when the number of
        virtual user is less than "1000"
16       And the "Transaction Response Time" should be less than "3" when the number of virtual
        user is less than "1000"
17       When the "Processor" is monitored
18       Then the "% Processor Time Counter" should be less than "70" when the number of
       virtual user is less than "1000"
```

Figure 5.4: Snippet of textual representation of the Changepoint Canopus Performance Monitoring model

5.3.3 Canopus Performance Scenario

The Changepoint usage scenarios and workload distribution were defined based on the application requirements, describing the user profiles and their respective interactions. Figure 5.5 shows part of the graphical representation of the `Canopus Performance Scenario` model for the Changepoint application (full models designed during this case study can be found in Appendix B). This model contains 7 user profiles: `Business Administrator`, `Development Team`, `Post Sales`, `Pre Sales`, `Project Manager`, `Resource Manager` and `Solution Architect`. There are 63 script objects associated with these user profiles, which represent the behavior of each user profile when interacting with a system functionality. Moreover, each association relationship, between a script object and a user profile, has a percentage that defines the number of VU, from the total number of users defined in the workload model, *e.g.* 96.1%.

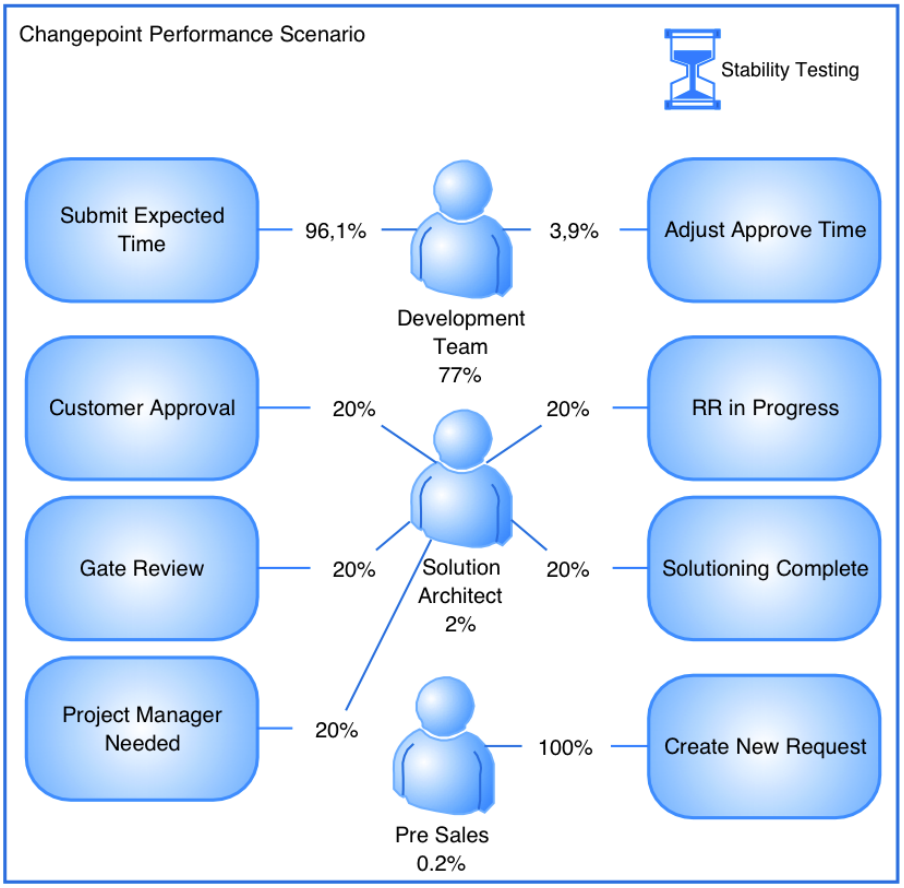


Figure 5.5: A partial graphical representation of the Changepoint `Canopus Performance Scenario` model

The user profile total number of VU is defined by the percentage annotated in the user profile element in this model. For instance, the `Development Team` user profile represents 77% of the workload of an entire Changepoint scenario. This workload is applied to execute the `Submit Expected Time` script based on the `Stability Testing` workload object. This object is decomposed in a `Canopus Performance Workload` model, depicted in Figure 5.6. This workload model

defines that 1000 VU will be running over Changepoint during 4 hours (test duration time). Besides, this model also presents information about the users ramp up and ramp down. For instance, during the test execution 100 VU will be added during every 10-minute interval. In the same way, the ramp down defines the way in which the VU will leave the SUT. Thus, the `Stability Testing` workload defines that 1000 VU will be simultaneously executing during 2h10min, the users ramp up time 1h40min and users ramp down time 10min therefore, completing 4 hours of test duration time. For instance, our `Development Team` user profile will execute 77% of all amount of VU (see Figure 5.7) from of 1000 VU (line 5), *i.e.* 770 (line 12). Thus, this user profile has association with two scripts: `Submit Expected Time` with 96.1%, *i.e.* 740 VU (line 14) from 770 VU this user profile.

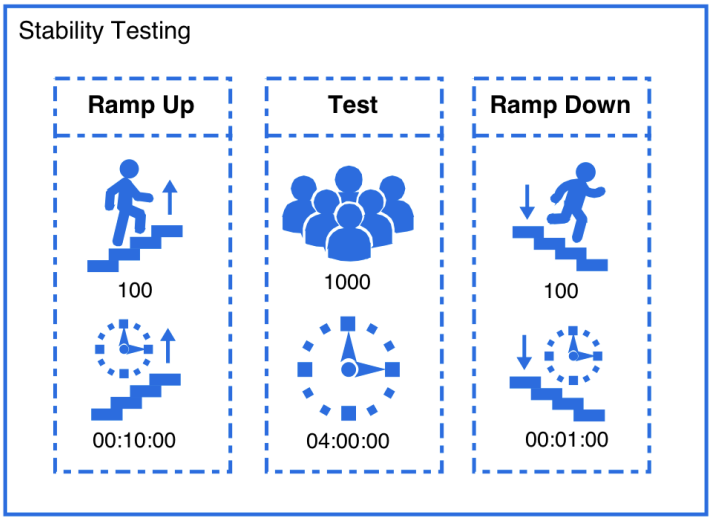


Figure 5.6: Graphical representation of the Changepoint `Canopus Performance Workload` model

A textual representation of the graphical performance test scenario, depicted in Figure 5.5 and Figure 5.6, is presented in Figure 5.7. This textual representation, as mentioned before, is structured based on the Gherkin language, but with elements that represent the semantics for performance testing, *e.g.* virtual users, ramp up and ramp down. It is worth to highlight that in this textual representation the percentage distribution among VU is already expressed in terms of values based on the workload.

5.3.4 Canopus Performance Scripting

As presented in Section 4.6, it is possible to associate a subgraph model for each script element from a `Canopus Performance Scenario` model. Therefore, each script element presented in the Changepoint scenario model is decomposed into a `Canopus Performance Scripting` model, which details each user interaction with the SUT functionalities. Figure 5.8 presents a snippet of the `Submit Expected Time` script. This model is composed of four activities, each one of them has another subgraph model associated, represented by "+" (plus) on the bottom script element. This decomposition allows reusing part of the modeled performance scripts. For instance, in this

```
1   Feature: Execute the test scenario for a workload.

2  Scenario: Evaluate the #Stability Testing# workload for "1000" users simultaneously
3     Given "100" users enter the system every "00:10:00"
4       And "100" users leave the system every "00:01:00"
5       And "1000" users register into the system simultaneously
6       And performance testing execution during "04:00:00"
7      When "1%" ("10") of the virtual users execute the #Business Administrator# user profile
       :
8      Then "10%" ("1") of them execute the #Partner Expense Association# script
9       And "30%" ("3") of them execute the #Order Association# script
10      And "30%" ("3") of them execute the #Order Unprocessing# script
11      And "30%" ("3") of them execute the #View Performance Dashboard# script
12     When "77%" ("770") of the virtual users execute the #Development Team# user profile:
13     Then "3.9%" ("30") of them execute the #Adjust Approve Time# script
14      And "96.1%" ("740") of them execute the #Submit Expected Time# script
15     When "0.8%" ("8") of the virtual users execute the #Post Sales# user profile:
16     Then "50%" ("4") of them execute the #Update Request Fields# script
17      And "50%" ("4") of them execute the #Upload Signed Documents# script
18     When "0.2%" ("2") of the virtual users execute the #Pre Sales# user profile:
19     Then "100%" ("2") of them execute the #Create New Request# script
20     When "1%" ("10") of the virtual users execute the #Resource Manager# user profile:
21     Then "10%" ("1") of them execute the #RM Configure Stream And Type Report# script
22      And "10%" ("1") of them execute the #RM Configure Stream Variant# script
23      And "20%" ("2") of them execute the #RM View Stream Roles Skill Items# script
24      And "20%" ("2") of them execute the #RM Export Skills And Streams# script
25      And "20%" ("2") of them execute the #RM Import Skills And Streams# script
26      And "20%" ("2") of them execute the #Select RM Request# script
27     When "2%" ("20") of the virtual users execute the #Solution Architect# user profile:
28     Then "20%" ("4") of them execute the #Customer Approval# script
29      And "20%" ("4") of them execute the #Gate Review# script
30      And "20%" ("4") of them execute the #Project Manager Needed# script
31      And "20%" ("4") of them execute the #RR In Progress# script
32      And "20%" ("4") of them execute the #Solutioning Complete# script
```

Figure 5.7: Snippet of textual representation of the Changepoint `Canopus Performance Scenario` model

case study a `_Login` script element is decomposed into a model that is included into several others scripts.

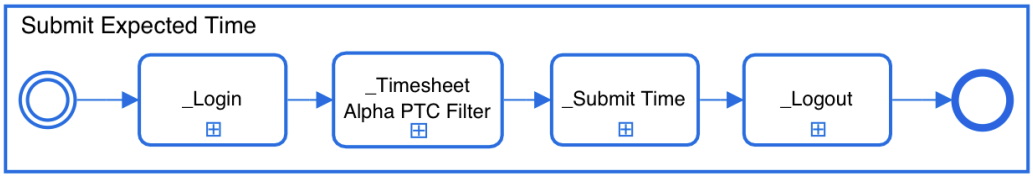


Figure 5.8: Graphical representation of the Changepoint `Canopus Performance Scripting` model for the `Submit Expected Time` script

The textual representation of the Changepoint `Canopus Performance Scripting` model for the `Submit Expected Time` script is shown in Figure 5.9. This is based on performance information annotated into the graphical representation from Figure 5.8. Since each activity from the graphical representation has a subgraph model associated, the interval of lines three to six present

a description of their respective subscripts. Exploring each one of them, it is possible to see details of their scripts, as presented in Figures 5.10 and 5.11.

```
1    Feature: Execute the performance test script for different user profiles.

2     Script: book submit time based on Test Case N.1 from #Submit Expected Time#
3  SubScript: based on Test Case N.1 from #_Login#
4  SubScript: based on Test Case N.1 from #_Timesheet Alpha PTC Filter#
5  SubScript: based on Test Case N.1 from #_Submit Time#
6  SubScript: based on Test Case N.1 from #_Logout#
```

Figure 5.9: Snippet of textual representation of the Changepoint Canopus Performance Scripting model for the Submit Expected Time script

Figure 5.10 shows the Canopus Performance Scripting model of the Timesheet Alpha PTC Filter script from the main model presented in Figure 5.8. This model is designed with 15 activity elements, 2 data tables (Server.dat and TaskId.dat) elements, a think time element (Clock element), and a couple of join and fork elements, *i.e.*, the Cal JQ, Time Sheet Status, Cal JQ Time Sheet Status and Form Time Sheet are executed in parallel. In some activities, test data must be dynamically generated. These data can be parametrized using data table elements. For instance, the TaskId.dat element provides test data for four activities, *e.g.* Time Sheet Open Task.

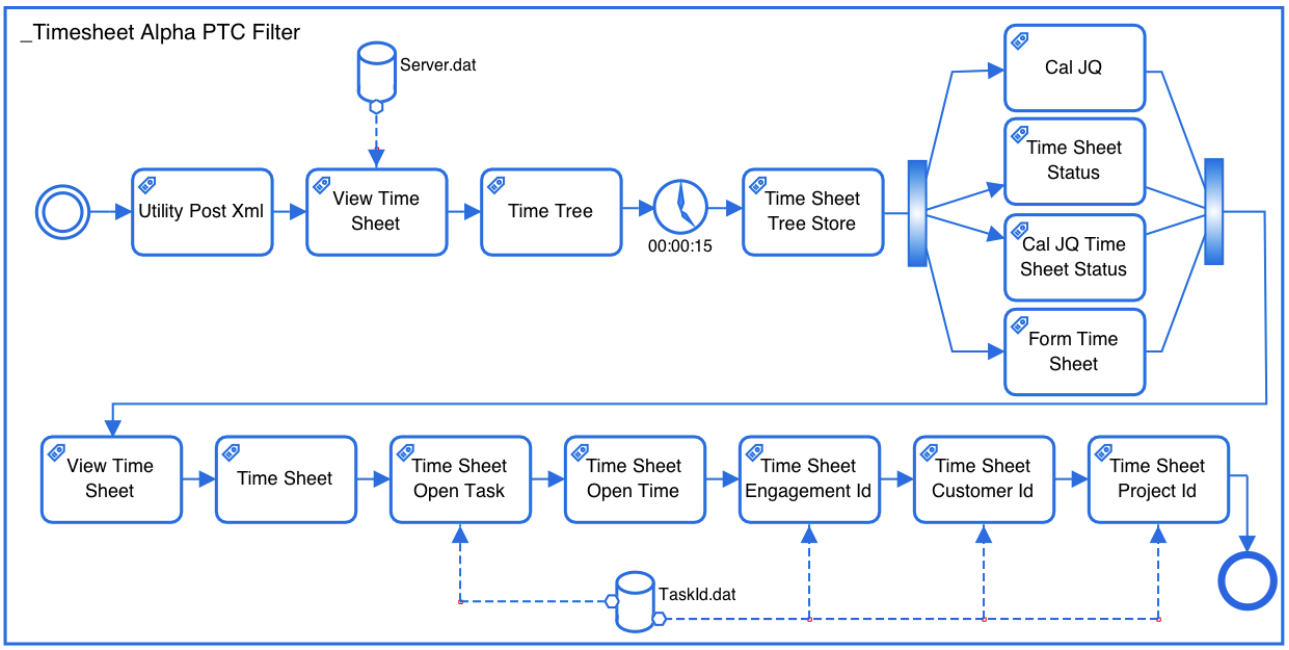


Figure 5.10: Graphical representation of the Changepoint Canopus Performance Scripting model for the _Timesheet Alpha PTC Filter script

A textual representation of the performance scripting model from Figure 5.10 is shown in Figure 5.11. An advantage of using a textual representation is to present all performance testing information annotated in the model, facilitating the view of all relevant information. Although

```
Feature: Execute the performance test script for different user profiles.

 Script: performance test script based on Test Case N. 1 from "Timesheet Alpha PTC Filter"
  Given the #Utility Post Xml# activity through "http://{Server}/Utility/UtilityPostXml.aspx
    ?rid={{rId}}&sno={{sno}}&ui=W&Action=0" action
   When the system randomize the "Server" data within {Server.Server}, which is dynamically
    generated and updated on "Each Interaction" based on a "Random" strategy
    ...
   Then I will be taken to "http://{Server.Server}/Core/TimeSheet/vwTimeSheet.asp?rid={{rId
    }}&sno={{sno}}&ui=W" action in the #View Time Sheet# activity
    And the system loaded the "rid" data values previously stored
    And the system loaded the "sno" data values previously stored
    And I need to wait thinktime during "00:00:15"
    ...
   Then I will be taken to "http://{Server.Server}/Projects/popCalJQ.asp?rid={{rId}}&sno={{
    sno}}&ui=W" action in the #Cal JQ# parallel activity
    ...
   Then I will be taken to "http://{Server.Server}/core/time_Sheet/frmTimeSheet_Status.asp?
    rid={{rId}}&reId=&sno={{sno}}&ui=W" action in the #Time Sheet Status# parallel activity
    ...
   Then I will be taken to "http://{Server.Server}/Projects/popCalJQ_TimesheetStatus.asp?rid
    ={{rId}}&sno={{sno}}&ui=W" action in the #Cal JQ Time Sheet Status# parallel activity
    ...
   Then I will be taken to "http://{Server.Server}/core/time_Sheet/frmTimeSheet.asp?rid={{
    rId}}&reId=&sno={{sno}}&ui=W" action in the #Form Time Sheet# parallel activity
    ...
   Then I will be taken to "http://{Server.Server}/Core/Treeviews/ifrtvTimeSheet.aspx?cid=
    tvEngTime&rid={{rId}}&reId=&sno={{sno}}&ui=W&searchFor=Perf&LoadFor=0&cptype=OpenTime&id
    =OpenTime&FirstLoad=1&view=PTC&render=alphabetic&showwbs=0" action in the #Time Sheet
    Open Time# activity
    And The system randomize the "Server" data within {Server.Server}, which is dynamically
    generated and update on "Each Interaction" based on a "Random" strategy
    ...
   Then I will be taken to "http://{Server.Server}/Core/Treeviews/ifrtvTimeSheet.aspx?cid=
    tvEngTime&rid={{rId}}&reId=&sno={{sno}}&ui=W&searchFor=Perf&LoadFor=0&cptype=eng_u_ex&id
    ={EngagementId}&FirstLoad=1&view=PTC&render=alphabetic&showwbs=0" action in the #Time
    Sheet Engagement Id# activity
    And The system randomize the "TaskId" data within {TaskId.TaskId}, which is dynamically
    generated and update on "Each Interaction" based on a "Random" strategy
    ...
    And The system randomize the "Engagement Id" data within {TaskId.EngagementId}, which is
     dynamically generated and update on "Each Interaction" based on a "Same" as "TaskId"
    strategy
    ...
Examples: Server.dat
| Server            |
| ausdwebsrv66      |
| ausdwebsrv69      |
Examples: TaskId.dat
| CustomerId        | EngagementId      | ProjectsId        | TaskId            |
| C8DDF527-4A13     | 9E988BD0-AD4D     | FA13E2DC-FE3E     | E9DD7653-13BB     |
| C8DDF527-4A13     | 555549F3-3473     | E8453F8B-F8B7     | C8AFD538-A7C7     |
```

Figure 5.11: Snippet of textual representation of the Changepoint Canopus Performance Scripting model

the graphical representation provides a better way to represent the main domain concepts, there are some information that are better represented in the elements from the textual representation. Furthermore, some activities (see Figure 5.11) have a dynamic parameter that refers to a data table, such as the `Time Sheet Engagement Id` activity (line 22) that refers to the `{EngagementId}` parameter, which in turn refers to a data table presented at the end of the script (line 25). Note that the test data associated to the `{EngagementId}` parameter will be updated on each interaction of a VU based on the `{TaskId}` parameter (line 23). It is worthwhile to highlight that in the textual representation the variability of flows for each test case (different paths through a graph and its subgraphs) is solved and is presented as a linear set of activities.

5.4 Case Study Analysis

We investigated and answered each one of the research questions stated previously in Section 5.3.1, based on the results of our case study and interviews conducted with a performance testing team. The performance testing team were 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[1] to symbolize performance elements that compose each Canopus Performance metamodel (Scripting, Scenario, Monitoring). rammstein

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 [Lik32]: 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.

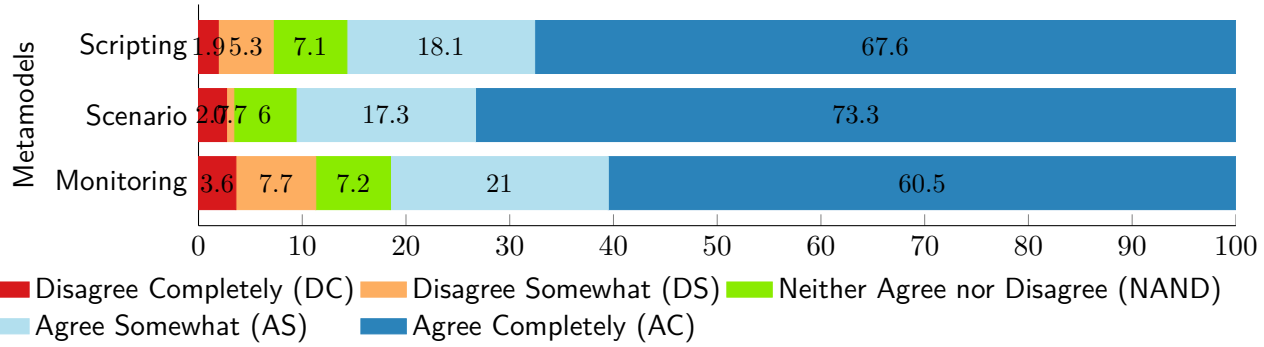


Figure 5.12: Frequency diagram of the graphical elements that compose Canopus, grouped by metamodel

The answers were summarized in the frequency diagram shown in Figure 5.12. The numbers in this figure are based on the evaluation of 37 elements: 13 elements for the `Canopus`

[1]Representativeness: regards to how much the graphical elements proposed by our DSL represent the performance testing domain.

`Performance Scripting` metamodel, 10 for `Canopus Performance Scenario` metamodel and 14 for `Canopus Performance Monitoring` metamodel. The frequency diagram presents the results grouped by each set of elements evaluated for each metamodel. As can be seen in Figure 5.12, 81.5% (60.5% AC + 21% AS) of the answers agree that the Monitoring elements are representative for the `Canopus Performance Monitoring` metamodel. For the `Canopus Performance Scenario` 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 `Canopus Performance Scripting` metamodel. These results are used as part of the evaluation of Canopus.

Each of the research questions mentioned in Section 5.3, are answered next.

RQ1. *How useful is it to design performance testing using a graphical DSL?* 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. *How intuitive is a DSL to model a performance testing domain?* 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 spent time on performance testing modeling activity.

5.5 Lessons Learned

In summary, the main lessons we have learned from the application of our model-based performance process, as well as the use of Canopus in an industrial case study are:

LL1) *DSL over a General Purpose Language (GPL).* A DSL can express better semantic of a domain than an adapted GPL, *e.g.* UML profile. In our previous works [RSO+14] [RBC+15], we empirically investigated the advantages and disadvantages on using an MBT approach or a CR approach. The results provided evidence towards proposing our DSL, since the UML approach that was applied did not completely support the entire domain concepts and rules. We are aware that

we must conduct further investigation to discuss the advantages on using a DSL instead of UML or other GPL on the performance testing domain;

LL2) *Global Software Development.* The GSD [CR09] refers to software development geographically, remotely or globally distributed, which aims to streamline the process of product development. In such scenario, it is common that infrastructure and performance teams are located in different countries. For this reason, it is important to adopt a standard language for creating scripts and models for performance testing, hence, we chose the English as default for the textual representation of Canopus, implicitly we avoid a cacophonous language [Fow10];

LL3) *Graphic design.* During the development of Canopus, we had some problems to design our graphical elements that compose our DSL. This drawback is due to lack of graphical design skills of developers. Also worth to note that the Language Workbench Meta Edit+ [Met15] aided to develop Canopus is compatible just SVG (Scalable Vector Graphics) format, which limits and requires more ability to build the graphical vector images. However, the reached results were satisfactory since the quality of graphical elements is the same, without quality loss, even when it is scaled;

LL4) *Performance testing engagement.* The experience with our industrial partner points out that it is common to the performance team to engage only on the final steps of software development. Canopus brings the performance team to engage in early stages of the software development process;

LL5) *More reuse using models than scripts.* It is easier to reuse a Canopus model to create/ compose other models than when reusing scripts in a CR-based approach. Moreover, the use of a CR-based approach could limit the reuse of previously generated artifacts, inducing the tester to rewrite new scripts from scratch;

LL6) *A picture is worth a thousand words.* The power of a graphical representation to help understand the problem domain and the business knowledge among teams is more efficient than using a common document specification. To integrate an inexperienced performance engineer in the performance team sometimes it is necessary to spend time reading the entire document. For instance, depending on the complexity of the software project, its specification can have more than 300 pages, including all change requests, and the worst is that this information is common and can not be reused to automate.

5.6 Chapter Summary

This chapter presented the Canopus being applied in the context of a collaboration between our university and a TDL from an Information Technology company. Throughout an industrial case study, we discussed the process to integrate Canopus to model-based performance testing in the context of a real environment. Canopus may be used to design performance testing models. In turn, the model-based performance testing process may be considered to support the performance testing automation, from a DSL to generate performance scripts and scenarios for different load generators.

Analyzing the threats to the validity of our proposal, we understand that having applied Canopus to a single industrial case study addressing only one application, could be a threat to the conclusions of the viability of our DSL. We are aware that we must further investigate the suitability of Canopus to other real industrial scenarios. Moreover, the selection of an ideal case study, as well as the complexity and the size of the models designed during the study may not be representative to generalize the results. Nevertheless, to mitigate this threat, we interviewed several performance engineers, as well as selected a set of distinct software projects from our TDL partner to choose the most representative case study to evaluate Canopus. Furthermore, the SUT used here is an off-the-shelf tool, a typical of a broad category of project management systems, which encourages us to try finding relevant results based on quality analysis of our scenario.

Currently, we are working close to our partner, which is trying Canopus to other actual projects. This cooperation will give us some good insights on which elements should be improved, altered or even included in Canopus. It is worthwhile to highlight that only a snippet of the Canopus Performance models is presented in this chapter. Hence, full models designed during this case study can be found in Appendix B, as well as survey details can be found in Appendix C.

Next, Chapter 6 will present an experimental study for evaluating Canopus, and for providing evidence about the benefits and/or drawbacks when using UML or Canopus approaches for modeling performance testing.

6. EMPIRICAL EXPERIMENT

"No experiment is ever a complete failure. It can always be used as a bad example."

— Paul Dickson

6.1 Overview

Performance testing modeling is a relative new research field. Researches investigating how to apply models to document performance testing information essentially started to be reported in the last decade. Motivated by the lack of a standard to represent performance testing information, our research group, in collaboration with an IT company, proposed a UML approach and lately a DSL to support performance testing modeling (see Chapter 4). To support our partner company on the decision process to replace UML by a DSL, we conducted an experimental study to provide evidence about the benefits and drawbacks when using UML or DSL for modeling performance testing.

In this chapter, we report an *in vitro* experiment, where the subjects designed *UML models* and *Canopus Performance models*. 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*. In our experiment, we follow the process and best practices presented by Wohlin *et al.* [WRH+12], which describes how to execute experiments in Software Engineering.

Our results indicate that, for performance modeling, the effort using a DSL was lower than using UML. Our statistical analysis showed that the results were valid, *i.e.*, that to design performance testing models using our DSL is better than using UML. Despite all our expertise on performance testing modeling, there was a lack of knowledge about the benefits and drawbacks of using DSL or UML to design performance testing model. Hence, we understand that some of the limitations are due to UML being a General-Purpose Language (GPL). To the best of our knowledge, there is no work that shows that UML performs better, in performance testing, than a DSL. This

chapter is organized as follows. Section 6.2 presents some context related to the company in place, as well as motivations for the study. Section 6.3 introduces the experiment instruments and the test documents. Section 6.4 presents the experiment design and introduces our research questions. Section 6.5 describes the execution of the experiment, while Section 6.6 presents our analysis and interpretation of results. Finally, we summarize the chapter in Section 6.7.

6.2 Experiment Context

In the past years our research group in performance testing has been investigating, in cooperation with a Technology Development Lab (TDL) of a global IT company, novel approaches, languages, notations and tools to support performance testing (see Section 3.3). One of our main research focus is the automation of performance testing process, mainly through the application of Model-based Testing (MBT) [UL06] approaches and its related languages and modeling notations. In this context we have proposed and empirically evaluated several approaches [RdOC+15], notations [CCO+12] [dSdMRZ+11], languages [BZR+14] and tools [RBC+15].

Essentially, all of our research topics are aligned with the industrial needs of our partner company. For instance, our research on performance testing automation wants to cover all the phases of performance engineering to apply model-based performance testing, from models and notations to supporting tools, providing a complete MBT solution in accordance with the company's needs. To reach this goal, we initially proposed the use of annotations on UML diagrams to model performance testing. After that, we developed a tool [RBC+15] that accepts these models as input and then generates performance scripts and scenarios for third-party tools/load generators. As time progressed and the use/interest in our solutions increased, we received feedback from the testing teams reporting some issues related to the use of UML diagrams to model performance testing. Based on that, we decided to implement a DSL, henceforth Canopus (see Chapter 4), for the performance testing domain in order to replace the UML models as our performance testing standard modeling notation.

The decisions on the replacement of a technology, specially in an enterprise context, must be based on strong evidences. Therefore, to provide evidence about the benefits and drawbacks on the replacement of UML by Canopus, we designed and setup an empirical experiment in collaboration with the TDL of our partner company.

6.3 Experiment Instruments

In this section we briefly present the UML approach for modeling performance testing, and LimeSurvey and Moodle[1] used as Systems Under Test (SUT) during the experiment training and execution:

[1] https://www.limesurvey.org | https://www.moodle.org/

- **UML profile for performance testing** [RBC+15]: Allows the use of UML use case and activity diagrams annotated with stereotypes and tagged values to model performance testing. Our approach to model performance information into UML models relies on stereotypes[2] to annotate test scenario information into use case diagrams and the test case information into activity diagrams. The performance stereotypes are the following:

  - **PApopulation**: This stereotype has six tags (denoted by the *TD* prefix): *TDpopulation* represents the number of virtual users that will be accessing the application; *TDhost* represents the host where the application is executed (defined in all actors of the use cases diagram); *TDrampUpTime* defines how long it takes for all users to access the SUT; *TDrampUpUser* represents the rate at which the number of users enter the SUT; *TDrampDownTime* defines how long it takes for all users to leave the SUT; *TDrampDownUser* represents the rate at which the users leave the SUT;
  - **PAprob**: Defines the probability of execution for each existing activity;
  - **PAtime**: Its related tag, *TDtime*, represents the expected time to execute each use case;
  - **PAthinkTime**: The *TDthinkTime* tag denotes the time between the moment the activity becomes available to the user and the moment the user decides to execute it, for example, the time for filling a form before its submission;
  - **PAparameters**: Defines the tags for the input data that will be provided to the application when running the test scripts (*TDparameters*), the user action (*TDaction*) and the method[3] (*TDmethod*);

- **LimeSurvey**: Is an open source Web application that allows non-technical users to quickly create online question-and-answer surveys. In the context of this experiment the application will be used as an SUT for training the subjects in the use of the modeling languages. Thus, the experiment's subject must design UML diagrams and Canopus Performance models to represent several users answering a survey. A detailed description about the training activities can be found at the experiment repository[4];

- **Moodle**: Is an open source learning management platform that provides a customized virtual environment for educators and students. This application was used as an SUT during the execution of the experiment. Therefore, the subjects must design a set of models, using both notations, to simulate a professor creating an activity into a course and then a set of students assigned to this course loggin into the application and visualize the activity. A detailed description about the experiment execution activities can be found at the experiment repository.

[2] A detailed description can be found in [CCO+12].
[3] HTTP method, *i.e.*, GET or POST.
[4] http://tiny.cc/SAC-SVT

6.4 Experiment Design

In this section we define the experiment objective and introduce our research questions.

6.4.1 Objective

The goal of this experiment is to provide evidence about the effort (spent time), intuitiveness and effectiveness when using UML or Canopus to create performance testing models.

Purpose

The purpose of the experiment is to evaluate the effort and suitability to design performance testing models when using Canopus.

Research Questions

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 Canopus?*

Null hypothesis, H_0: effort is the same when using UML and Canopus to design a performance testing model.

Alternative hypothesis, H_1: the effort is lower when using UML to design a performance testing model than when using Canopus.

Alternative hypothesis, H_2: the effort is lower when using Canopus to design a performance testing model than when using UML.

RQ4. *How effective is it to design performance testing model when using UML or Canopus?*

RQ5. *How intuitive/easy is it to design performance testing model when using UML or Canopus?*

Summary of Definition

Analyze the design of annotated *UML models* and *Canopus Performance models*
for the purpose of *evaluation*
with respect to the *effort and suitability*
from the perspective of the *performance testers and the performance engineers*
in the context of *industry and academia environments*
for *modeling performance testing*.

6.4.2 Selecting and Grouping Subjects

We chose an *in-vitro* approach to avoid external influences during the execution of the experiment. Therefore, all activities executed by the experiment subjects were performed in a laboratory, under controlled conditions. After the definition of the experiment design (see Figure 6.1), we focused on the selection of the subjects, which is one of the most important activities in an experimental study. Thus, we spent a considerable effort on inviting and selecting subjects from industry and academia. First and foremost, we focused on the selection of subjects based on the information provided by the partner company about the target subject profiles: junior and senior performance analysts and testers. Therefore, we invited experienced performance engineers and testers from a local company and from a global IT company. We also invited undergraduate students from an university, in order to select subjects with non-industrial experience on performance testing. After selecting the subjects we set the dates to run the experiment sessions - a whole day for both training and execution sessions. Moreover, before the training session we asked the subjects to answer a background survey. From the data extracted from that survey, we randomly assigned the subjects into two groups (randomization). Furthermore, we also kept each group with the same number of subjects and with similar skills (balancing). During the execution session, each group started modeling with a different notation. While *Group 1* started modeling with UML, *Group 2* designed the models using Canopus. In the next phase, *Group 1* started modeling with Canopus and *Group 2* modeled using UML - all subjects executed both treatments (a paired comparison design). After all the subjects had executed the last experiment phase, they answered a post-experiment survey. It is important to highlight that we monitored the time spent by each subject to complete each phase. Thus, the time spent data and the survey results from all the subjects were used to draw the experiment conclusions.

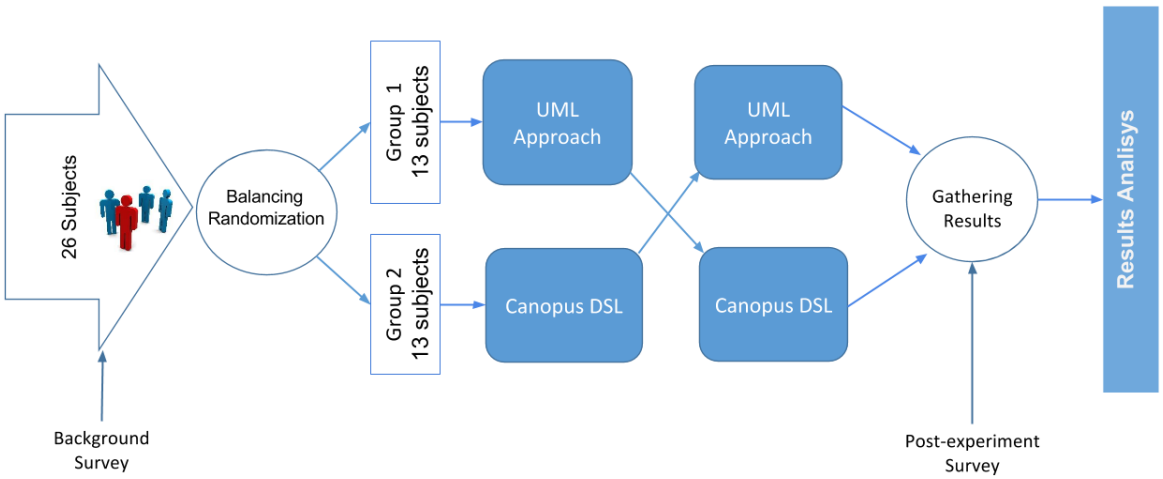


Figure 6.1: Experiment Design

The experiment design addressed according to general principles as follows:

Balancing: The subjects are randomly placed into each group (randomized block design), so that each approach is performed by the same number of subjects (UML or Canopus);

Blocking: The selected subjects for this experiment had different backgrounds in performance testing. Thus, to minimize the effect of those differences, the subjects were classified in two groups according to their skills in performance testing (inexperienced and experienced groups - see Table 6.1). To define if a subject was inexperienced or experienced, we applied a survey, prior to the experiment, to qualify the subject background on performance testing, software modeling with UML and professional experience with modeling performance testing (see Appendix E);

Randomization: The subjects were randomly allocated to each testing approach - UML or Canopus. Moreover, as all subjects executed all treatments (paired comparison design), we randomly defined their execution sequence;

Standard Design Types: The design type presented aims to evaluate whether the effort of UML and Canopus are different. Thus, it is necessary to compare the two treatments against each other. As defined in [WRH+12], the One Factor with Two Treatments design type must be applied. This factor is the performance testing modeling approach that will be used and the treatments are the UML and Canopus approaches. The response variable is measured on a ratio scale, *i.e.*, to allow us to rank the items that are measured and to quantify and compare the sizes of differences between them.

The context of the experiment is characterized according to four dimensions:

(1) **Process**: We used an *in-vitro* approach, since it refers to the experiment in laboratory under controlled conditions. This experiment is not an industrial software testing, *i.e.*, it is off-line;

(2) **Participants**: Undergraduate, master and doctoral students, performance testers and engineers;

(3) **Reality**: The experiment addresses a real problem, *i.e.* the differences in individual effort to model performance testing using UML and Canopus approaches;

(4) **Generality**: It is a specific context since Canopus used in this experiment is a specific DSL to attend the performance domain, however, Canopus can be extended to other test paradigms.

6.4.3 Instrumentation

The main objects of the instrumentation for our experiment are the performance testing models composed of performance scripts, scenarios and workloads, designed in accordance with both approaches (UML and Canopus) for testing the Moodle application. Furthermore, guidelines were

provided for supporting the subjects on the execution session, such as performance requirements, technical specification and use cases. Moreover, we used two tools to support each one of the approaches: Astah Professional [Ast15] version 6.9.0, for modeling the use case and activity diagrams when applying the UML approach, and; MetaEdit+ [Met15] version 5.1, for designing the graphs supported by the metamodels developed by Canopus [BZR+14].

In the training session (using LimeSurvey), the UML approach was introduced to the subjects through an oral presentation using videos to demonstrate how the approach was applied to a real case study. Additionally, we provided the subjects with a manual about UML modeling for performance testing and a detailed instruction on how to use Astah to design performance models. Similarly, Canopus was introduced to the subjects through an oral presentation, which we demonstrated how Canopus could be applied for modeling performance testing. We provided the subjects with a manual about Canopus and also a detailed instruction on how to use MetaEdit+ to design Canopus Performance models. After that, the subjects had to design a user interaction with a Web-based survey. Furthermore, the subjects could ask open questions for each approach or clarify how to use the tools to design the performance models.

In the execution session, the subjects interacted with a different Web application - Moodle. To execute the tasks that compose the experiment, the subjects were introduced to guidelines, describing in details the test specification. Figure 6.2 shows one of the use cases used in the experiment. Based on those documents, the subjects had to design the performance models, in accordance with the approach guidelines. Figure 6.3 presents the UML activity diagram designed in accordance with the Moodle use case described in Figure 6.2. Figure 6.4 presents a `Canopus Performance Scripting` model designed in accordance with the same specification. More details and complete specification of experiment instruments can be found in Appendix D.

Due to the heterogeneity of subjects sources, we decided to execute the experiment *in loco*, but *in vitro*, controlling the environment against external interferences. We collected effort metrics for each subject to answer our **RQ3**. To answer **RQ4** and **RQ5** we collected data from the post-experiment survey.

6.4.4 Threats to Validity

In this section, we describe the threats to the experiment validity, and how we work to mitigate them. We adopted the threat classification scheme published by [CC79], which is divided in four types of threats:

- **Conclusion validity**: Threats to the conclusion validity affect the power of our conclusion about the relations between the use of modeling languages and the results of the experiment. In this experiment context, the small number of subjects, in special the small group of subjects from industry, is a significant threat to the conclusion validity. Other threats to our experiment conclusion validity are the following: *Measures reliability*: this type of threat is related

```
Use Case: Add Activity
#Description: The assignment activity module enables a teacher to communicate tasks,
  collect work and provide grades and feedback to students.
#Actors: Teacher.
#Finality: Allow teacher to assign activity to course.
#Pre-Condition: The teacher has logged in the Moodle.
1. Select Course
     action: go to "http://www.cepes.pucrs.br/moodle/" where id equal 22
2. Enable Editing
     action: submit "Turn editing on" button
3. Click Add an Activity or Resource
     action: click on "Add an Activity or Resource" link
4. Select Assignment Option
     action: select on "Assignment" option and submit "Add" button
5. Click
     action: type "Name" text field and type "Description" text area
5.A. Save and Display
       action: submit "Save and Display" button
5.B. Save and Return to Course
       action: submit "Save and Return to Course" button
#Pos-Condition: Activity assignment in the course.
                Students able to upload yours answer.
```

Figure 6.2: A Moodle Use Case specification

to the researcher bias when analysing the experiment results. To mitigate this threat, the experimental results were independently validated by two researchers. Moreover, the analysis of quantitative data do not involve human judgment; *Random irrelevancies in the experimental setting*: this type of threat entangles our ability to see a relationship between the treatments and the experiment results. To mitigate this threat, all the experiment activities involving the subjects were executed in a laboratory, isolated from external influences (noise, interruptions, etc.). Moreover, the subjects were not allowed to use mobile phones or any other type of communication with the external world; *Random heterogeneity of subjects*: we selected

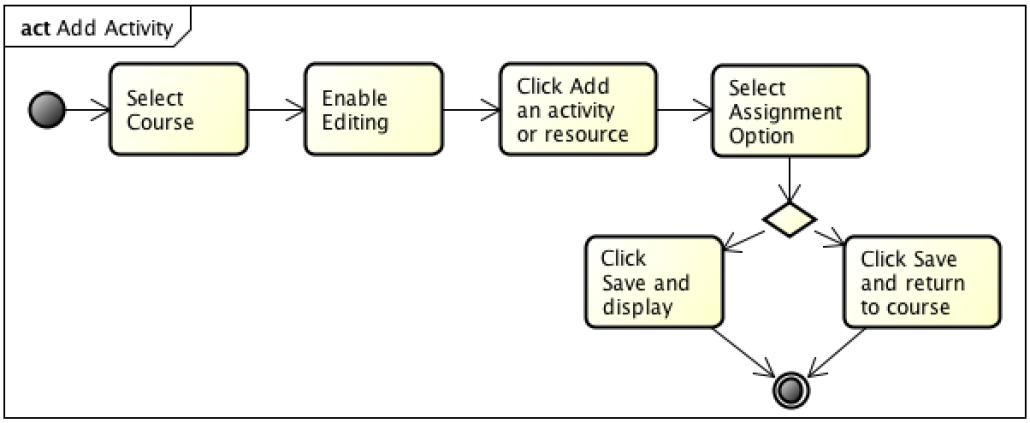


Figure 6.3: UML activity diagram of the Use Case specification from Figure 6.2

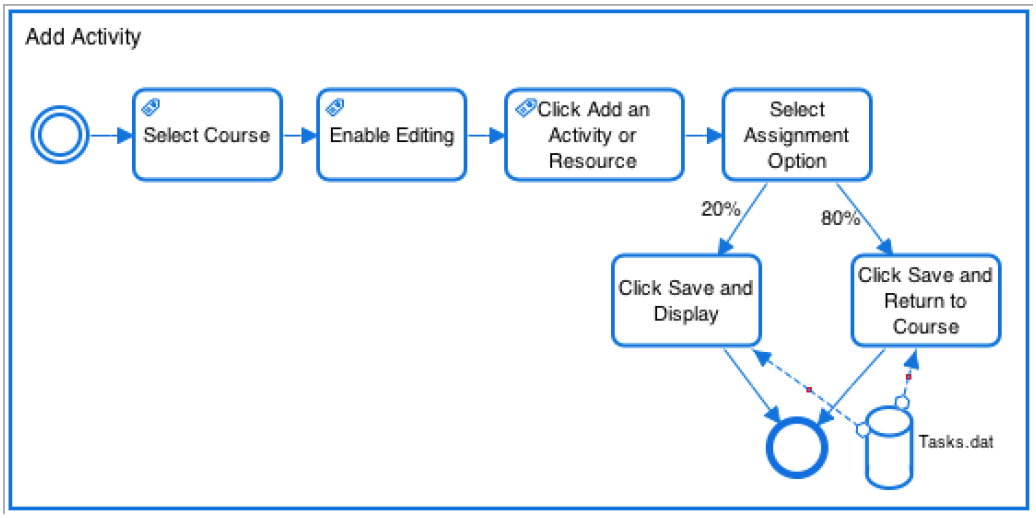


Figure 6.4: `Canopus Performance Scripting` of the Use Case specification from Figure 6.2

a diverse group of subjects: seven from a large IT company, six from a local company and thirteen undergraduate students. The selection of some subjects with no industrial experience on performance testing, and others with years of experience in software testing and modeling may be a threat to the validation of the experiment results. To mitigate this threat we defined experience on performance testing as blocking variables: inexperienced (IN) and experienced (EX);

- **Internal validity**: We identified the following threats to the internal validity of our experiment: *History*: we chose to perform the training and execution sessions in a single day. The definition of the date was chosen collaboratively with the subjects, to avoid defining a date in which they may be committed with others activities, *e.g.*, we did not execute the experiment in an exam period (inexperienced subjects) and close to the start/end of an important project (experienced subjects); *Maturation*: we ran the experiment in a single day. In the morning, we ran the training session. Then, the subjects had an interval to rest. In the afternoon, we ran the experiment execution session. We chose to split the experiment session to avoid overwhelming the subjects. Nonetheless, we chose to execute both sessions in a single day to avoid that subjects forgot what was presented in the training session; *Selection*: a survey was applied to assess the knowledge and the experience of subjects and then used to select and group (block) the subjects;

- **External validity:** We identified the following threat to the external validity of our experiment: *Subjects*: the selection of the subjects that may not be representative to the performance testing community is a threat to the external validity of our experiment. To mitigate this threat, we invited software testing professionals with different levels of expertise in performance testing from two companies. Our decision on inviting undergraduate students was made in order to provide data on the use of the languages for subjects with no knowledge/expertise in performance testing;

- **Construct validity**: A threat to the construct validity is that we use a single application as SUT, and the subjects modeled a limited set of test cases of the application (mono-operation bias).

6.5 Operation of the Experiment

This section discusses the preparation and execution steps performed during the experiment operation.

6.5.1 Preparation

Our preparation to the experiment execution includes: to identify what application requirements will be modeled during the training and execution sessions; to prepare the approach guidelines, and; to identify and create practical examples to show during the training session[5]. Furthermore, all laboratory computers were prepared with all the necessary documents, as well as the proper tools (Astah and MetaEdit+). Moreover, we also applied a pre-experiment survey to obtain background information on the subjects. Based on this information, we blocked the subjects and categorized them into two equivalent groups (balancing, randomization).

6.5.2 Execution

The experiment took place in June of 2015. During the training session, both approaches were applied, and for each approach two tasks are executed: performance scenario and scripting modeling.

The performance scenario specification of LimeSurvey is composed of two user profiles: a student and a professional respondent. The workload is composed of one thousand virtual users for a testing duration of four hours. This workload executes one script, named "Answer The Survey", which is composed of twelve activities that represent each one of the questions that comprises a survey. The performance script model is based on performance requirements to the SUT training.

Table 6.1: Assigning subjects to the treatments for a randomized design

Treatments	Blocks	Number of Subjects
UML	Inexperienced (IN)	13
	Experienced (EX)	13
Canopus	Inexperienced (IN)	13
	Experienced (EX)	13

[5]The experiment instruments are available at http://tiny.cc/SAC-SVT

Table 6.1 presents the distribution of experiment subjects between the treatments and their respective block variables. Hence, twenty-six subjects were assigned with two groups (13 subjects per group). Each group started the execution of one of the two treatments (UML or Canopus).

During the experiment execution session, which followed the same systematic approach applied in the training phase. Thus, the subjects must performed two tasks. One task was to design performance models using UML and another was to design performance models using Canopus. Each session is described next:

- **UML**: To design performance models in accordance to the experiment guidelines using UML profile for performance testing:
  - *Scenario Task*: This task consists of designing a UML model from scratch, based on the performance requirements, using Astah. The performance scenario model, represented here by a use case diagram, is composed of two actors: students and teachers. These actors are associated with three use cases, which will be detailed into activity diagrams;
  - *Scripting Task*: The experiment subject had to design the performance scripts using the modeling strategy described in Appendix D, aggregating performance test information to the model. Therefore, they had to design three activity diagrams to represent a test case describing the interaction between the user and the SUT;
- **Canopus**: To design performance models in accordance to the experiment guidelines using Canopus:
  - *Scenario Task*: In this task the subjects had to use Canopus to design the `Canopus Performance Scenario` in accordance with the provided guidelines. In order to model that, the experiment subject had to use MetaEdit+, which contains the Canopus Performance metamodels. It is important to highlight that `Canopus Performance Scenario` has elements that may be decomposed into other elements, *e.g.* `Canopus Performance Workload`, which is modeled to represent features such as test duration, numbers of virtual users, ramp up and ramp down;
  - *Scripting Task*: Similarly to Scripting Task from the UML approach, in this task the subjects had to model the `Canopus Performance Scripting` to represent the interaction between a virtual user (VU) and the SUT. Figure 6.3 shows the `Canopus Performance Scripting` designed in accordance with the Moodle use case specification described in Figure 6.2. Additionally, the entire task comprises two more diagrams that represent the *Sign In* and *View Activity* use cases.

6.6 Results

RQ3. *What is the effort to design a performance testing model when using UML or Canopus?* Table 6.2 presents the summarized effort data (time spent by subjects) to perform

each task using each approach, as mentioned in Section 6.5.2. In Table 6.2, the columns Scenario and Scripting present the average time per blocks and treatments, respectively. The Total column represents the sum of the previous columns (Scenario+Scripting). Based on the results summarized in Table 6.2, 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 6.2: 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			
Canopus	Inexperienced	10.54	39.31	49.85	10.23	40.85	51.08
	Experienced	9.92	42.38	52.31			

Table 6.3 shows the subject's effort data, in minutes, per task. A first glance at the table identifies the parting between inexperienced and experienced subjects, so using them as control variables (blocks) was important in our case. In Table 6.3, we grouped each task effort in one of three parts: 1) 29.5% had the best performance (first quartile); 2) 41.7% represent the median performance (median) from all subjects; and, 3) 28.8% had the worst performance (third quartile).

On one hand, as can be seen in Table 6.3, most of the subjects that performed worse, *i.e.* 66.7%, were among the ones blocked as inexperienced subjects. Naturally, some of the subjects did not have the expected results due to their skills and knowledge. For instance, subject IN05 from inexperienced had one of the best performance from all subjects; also, subjects IN07 and IN11 had some of the worst performances in this block. In turn, subject EX04 had the best performance of both treatments. Also, subject EX01 from experienced group had one of the worst performances. This could be expected, since we do have subjects that are above the average in several aspects, and also an experienced subject that could not yet have the experience necessary to achieve good results as a whole. Such a result was foreseen as a threat for our experiment as explained in Section 6.4.4.

On the other hand, from experienced subjects, 80.8% were among the subjects that we classified in the first and median quartile, and only 19.2% are in the third quartile (worst performance). Experienced subjects from UML approach had their results concentrated in the third quartile (10.3%). We infer that experienced subjects from UML approach had improved their results when comparing with inexperienced, which had major results in the third quartile while experienced subjects concentrated their results in the first and median quartile with 89.7%. Possibly, these results had influences from previous skills and knowledges about UML (see Figure 6.7). Inexperienced subjects were distributed nearly equally among the three parts, respectively, 29.5%, 32%, and 38.5%.

Figure 6.5 depicts the Box-Plot graph of the Scenario Task data set, represented by UML_{Scen} and DSL_{Scen} 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 was 5.02

Table 6.3: Effort data per subject (minutes)

		Treatments					
		UML			Canopus		
		Blocks					
Blocks	**Subject**	**Scenario**	**Scripting**	**Total**	**Scenario**	**Scripting**	**Total**
Inexperienced	IN01	19	37	56	11	37	48
	IN02	12	47	59	11	40	51
	IN03	15	50	65	7	40	47
	IN04	21	39	60	10	35	45
	IN05	12	33	45	8	33	41
	IN06	9	70	79	17	51	68
	IN07	20	60	80	15	45	60
	IN08	7	68	75	5	23	28
	IN09	9	51	60	8	50	58
	IN10	20	54	74	10	37	47
	IN11	20	60	80	15	45	60
	IN12	25	75	100	10	40	50
	IN13	15	40	55	10	35	45
Experimenced	EX01	10	62	72	17	51	68
	EX02	5	26	31	15	52	67
	EX03	12	41	53	9	39	48
	EX04	10	40	50	8	37	45
	EX05	11	39	50	5	32	37
	EX06	14	48	62	7	53	60
	EX07	7	41	48	7	39	46
	EX08	18	60	78	15	41	56
	EX09	14	53	67	8	43	51
	EX10	12	51	63	9	38	47
	EX11	9	49	58	10	38	48
	EX12	15	53	68	9	49	58
	EX13	13	55	68	10	39	49
First quartile ($\leq$)		10	40.25	55.25	8	37	46.25
Median ($\lozenge$)		12.5	50.5	62.5	10	39.5	48.5
Third quartile ($\geq$)		17.25	58.75	73.5	11	45	58

Legend - **Hashed cells**: *first quartile*; **Blank cells**: *median*; **Dotted cells**: *third quartile*

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 6.5 also presents the Box-Plot graph of the Script task. This task is represented by UML_{Scr} and DSL_{Scr} 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 standard deviation 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 6.5 also shows the Box-Plot graph of the summarized data set, *i.e.*, the sum of Scenario and Script 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 standard deviation (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.

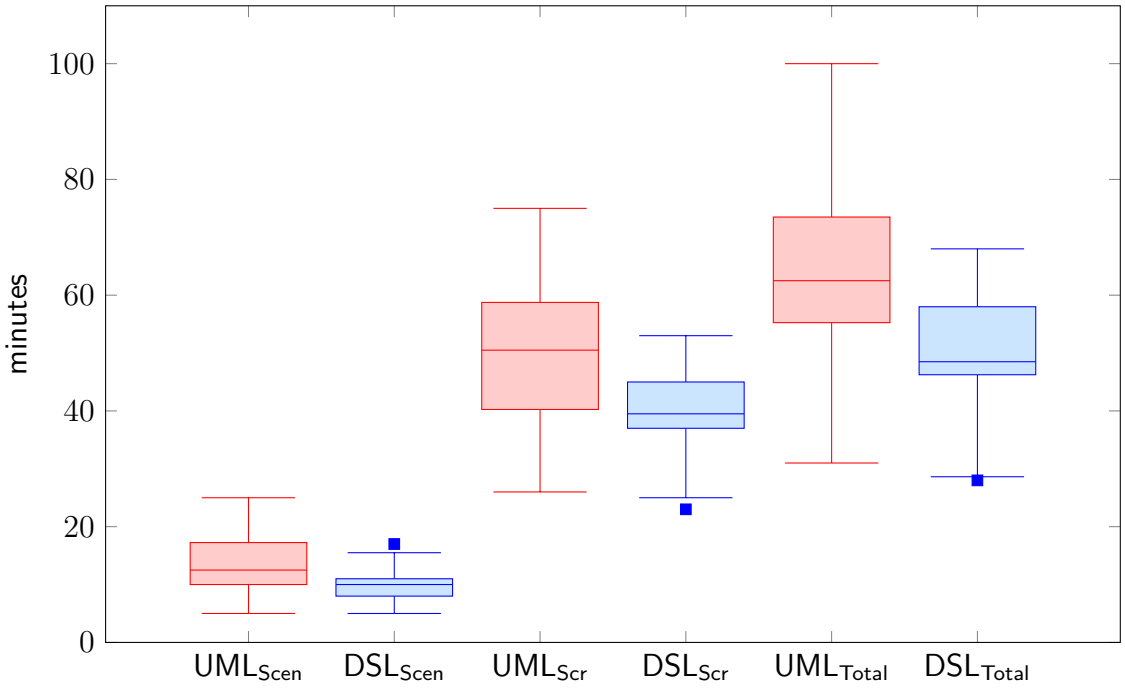


Figure 6.5: Boxplot - treatments per task

Figure 6.6 presents the box-plot graph grouped by blocks. The medians of execution times with UML_{IN}, UML_{EX}, DSL_{IN} and DSL_{EX} were, respectively, 65, 62, 48 and 49 minutes. Moreover, the standard deviation for each block was, respectively, 14.65, 12.53, 10.11 and 9.0.

It is worthwhile to highlight that of all the data sets analyzed, the standard deviations of Canopus presents their values smaller than UML, *e.g.*, it gives us an idea of how close their data sets are, respectively, to the average value. How much more precise and tightly grouped the data set is. This is better from a statistical point of view.

As for hypothesis testing, we used the PortalAction statistical package [Por15] integrated with MS Excel to test our hypothesis from the collected data sets. We performed the Kolmogorov-Smirnov [Por15] test to verify the normality of data distribution (Table 6.4). 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. The Scenario data set, which showed a *p*-value (0,002396184) lower than α. For this reason, we assumed that the distribution was not normal. Therefore, we applied a non-parametric test: Wilcoxon [Por15] signed rank test. We applied a non-parametric test since it uses the median instead of average as used in the parametric test. This solves the problem with outliers. For each data set (Scenario, Scripting and Total), we applied the statistical test to the paired samples, *i.e.* to compare the effort spent to model performance using UML or Canopus (**RQ3**). As presented in Table 6.5, for all samples pairs, the results of the Wilcoxon test reject the H_0 hypothesis. Therefore, we can conclude that there is, statistically, a noticeable difference in effort to design a performance testing model when using UML and Canopus. Thus, for all data sets we confirmed the alternative hypothesis H_2.

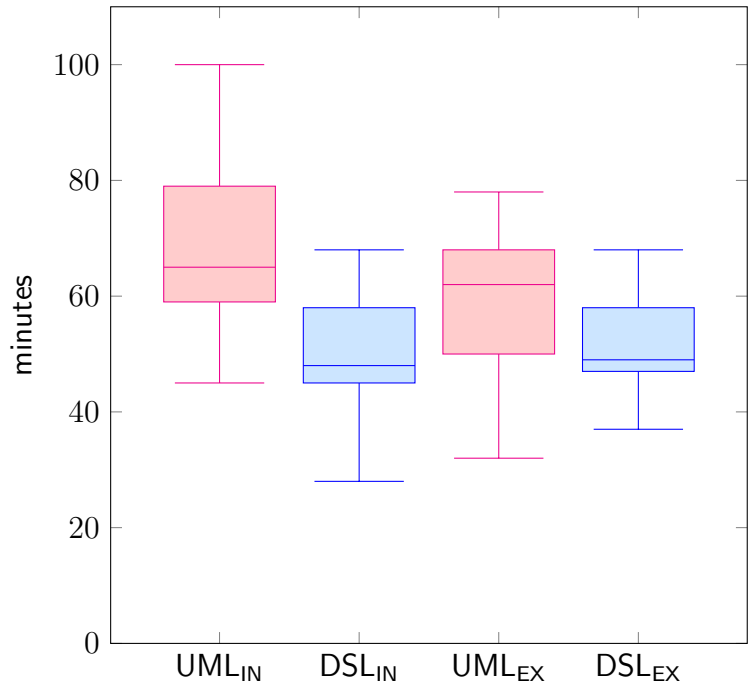


Figure 6.6: Boxplot - treatments per block

The major concern of the experiment is the fact that we made the experiment on only one system, Moodle, and with few people, statistically, not enough. Despite of that, the statistical analysis demonstrated that the sample is normal and relevant.

Table 6.4: Kolmogorov-Smirnov normality test

Treatment	**Scenario**		**Scripting**		**Total**	
	Test	***p*-val**	**Test**	***p*-val**	**Test**	***p*-val**
UML	0,126	0,355	0,124	0,377	0,086	0,886
Canopus	0,219	0,002	0,162	0,074	0,157	0,098

Table 6.5: Wilcoxon signed rank test results

Treatment	**Scenario**	**Scripting**	**Total**
UML/Canopus	0.007055169	0.000603644	0.000124493

RQ4. *How effective is it to design a performance testing model when using UML or Canopus?* **RQ5.** *How intuitive/easy is it to design a performance testing model when using UML or Canopus?*

After designing the performance models using both approaches, the subjects answered a survey composed of:

(a) Assertions to answer how they rate with their technical knowledge in their different backgrounds according to the context of the experiment, based on a four points scale: "Low, no prior knowledge"; "Regular, read a book or followed a course"; "Average, some industrial experience (less than 6 months)"; and, "High, industrial experience";

(b) Statements to survey how much they concur with our Canopus features, based on a five points Likert scale [Lik32]: Strongly Disagree (SD), Disagree (D), Neither Agree nor Disagree (NAD), Agree (A) and Strongly Agree (SA);

(c) Open questions to extract their opinions.

As shown in Figure 6.7, most of the experiment subjects have litle knowledge and few skills of technical issues, where most of them answered that they have more technical knowledge, having chosen the "Regular, read a book or followed a course" option, influenced by their university time. Despite the experiment subjects having had more skills and knowledge of UML than other performance modeling languages, or DSL, it did not influence the achieved outcomes, which corroborated that Canopus approach is more productive than UML.

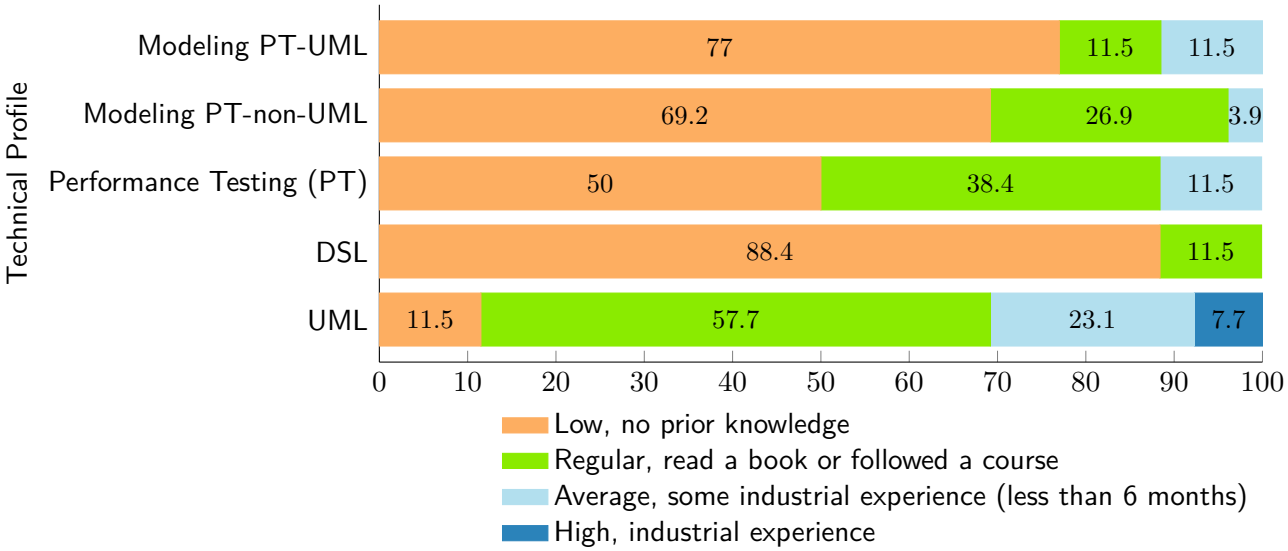


Figure 6.7: Frequency diagram of the profile experiment subjects

As can be seen in Figure 6.8, the statement most accepted is that Canopus has **Expressiveness**[6] (61.5% SA, 27% A and 11.5% NAD), followed by that it has **Representativeness**[7] (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 with respect to **Intuitiveness**[8] (53.9% A, 26.9% NAD and 19.2% D).

The main advantages on the use of the UML approach, by the subjects point of view were: (a) *It was easier because I already had some experience with UML*; (b) *I already had used Astah tool to design UML, so it was easier to use*; (c) *There are tons of documentation and discussion groups on the Internet*. The subjects also pointed some disadvantages on the use of UML: (a) *Lack*

[6]Expressiveness: defines the quality of graphical elements composed by visual variables (color, size, brightness, shapes, and textures) used in a notation for supporting the usability of graphical modeling languages.

[7]Representativeness: regards to how much the graphical elements proposed by our DSL represent the performance testing domain.

[8]Intuitiveness: regards to how much the graphical elements proposed by Canopus works the way the user does what it should do enough way through intuition without any training or rational thought is necessary, feeling to do it naturally. It means something that can be naturally and instinctively understood and comprehended.

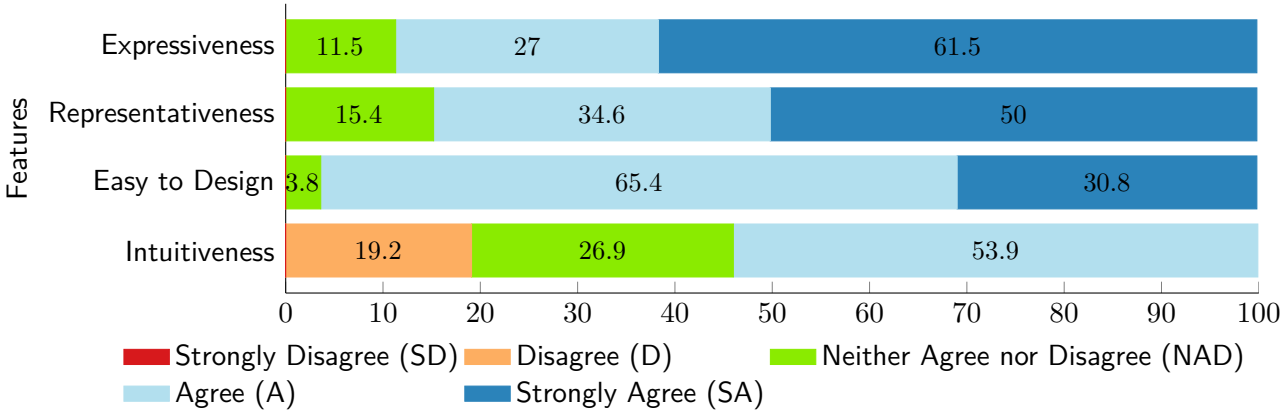


Figure 6.8: Frequency diagram of the Canopus

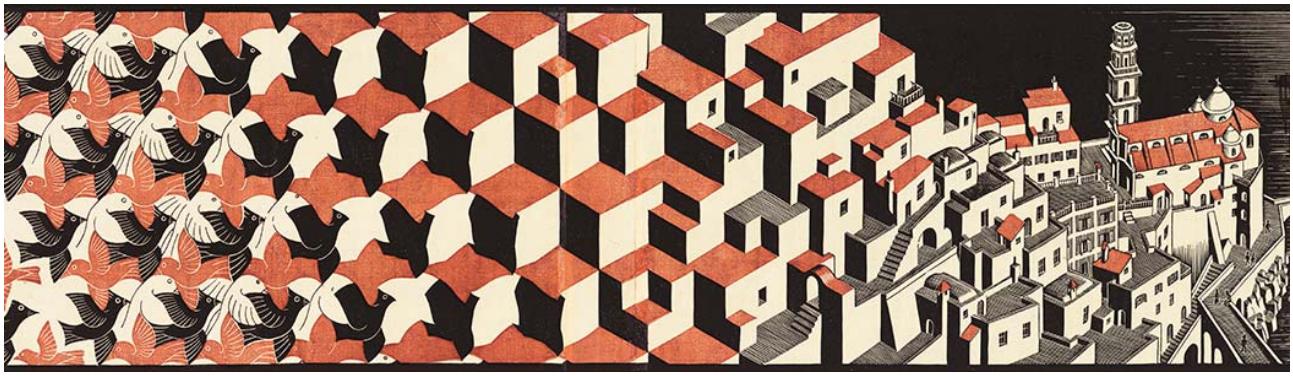
of interaction/reuse of models with parameters; (b) *Higher probability of making a syntax error when typing the tag values*; (c) *Some performance testing information cannot be easy to design with UML*.

Similarly, the subjects reported advantages on the use of Canopus: (a) *Modeling was simple and intuitive. Promotes the code reuse in all modeling phases. Easy and fast understanding of the interface*; (b) *It is intuitive to realize the relations that are being performed. The scenario creation is intuitive and the probability of making a syntax error is smaller*; (c) *Very intuitive. Allows to add many performance testing information to the models. Adding parameters is much easier*. The subjects pointed out the following disadvantages: (a) *Learning curve - at the beginning it was complex to use. It was difficult to understand which element should be used*; (b) *Lack of mechanism to duplicate an element*.

6.7 Chapter Summary

In this chapter, we have presented an *in vitro* experimental study for evaluating our DSL, and for providing evidence for the benefits and/or drawbacks when using UML or DSL approaches for modeling performance testing. The experimental results indicate that the use of Canopus reduces the effort required to create models when compared to UML. Moreover, the experiment subjects pointed out that Canopus has more expressiveness and representativeness, and is easier to design than UML. Hence, these findings can provide support to our partner company in the decision process to replace UML by Canopus for modeling performance testing. We are aware that the sample size that was used to base some of our conclusions is not geographically relevant. Moreover, a group of subjects did not have an adequate experience level or knowledge on performance testing. Therefore, we designed our experiment protocol with the intention to replicate it in the future to collect more results. Furthermore, based on the achieved findings, testimonials from experiment subjects and our lessons learned to conduct this experiment, we intend to improve our DSL, ensuring the evolution of Canopus for a new version of graphical and textual language, as well as new features.

7. FINAL REMARKS



"It's more fun arrive to a conclusion than to justify it."

— Malcolm Stevenson Forbes

7.1 Overview

It is well-known that the testing phase is one of the most time-consuming and laborious phases of a software development process [YHL+08]. Depending on the desired level of quality for the target application, and also its complexity, the testing phase can have a high cost. Normally defining, designing, writing and executing tests require 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 relevant 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) [UL06].

Although MBT is a well-defined and applied technique to automate some testing levels, it is not entirely explored to test non-functional requirements of an application, in particular, performance testing. One strategy to automate the testing process for a single testing level would implement a Domain-Specific Language (DSL) [Fow10] to deal with this particular problem domain. DSL can provide a graphical, textual or hybrid language to instantiate models based on their metamodels designed. Therefore, it would be relevant to develop a modeling language for the performance testing domain in order to expand the domain-specific testing as an efficient solution. By doing this, domain experts could express abstract models or languages to generate testing artifacts automatically, focusing on transforming the domain knowledge to improve the performance testing process. In this thesis, we proposed Canopus, which aims providing a graphical and textual DSL to

support the design of performance models and which can be applied in a model-based performance testing approach.

In this context, this thesis has contributed to theoretical and empirical studies to advance the performance testing area by applying MBT using a domain-specific language to support the design of performance modeling. The results have provided evidence for a positive answer to our general research question proposed in Chapter 1, *i.e.*, "*How to improve model-based performance testing using the domain-specific language in Web applications*" Section 7.2 revisits the achieved thesis contributions that support answering such a question. Section 7.3 summarizes the study limitations, and also sketches ongoing research and planned future work. Finally, Section 7.4 describes the academic contribution of the author in terms of publication.

7.2 Thesis Contributions

This section states the achievements of this thesis as follows.

Canopus - a domain-specific language for modeling performance testing: We proposed, designed and developed Canopus (Chapter 4). Canopus is a graphical and textual language to model performance testing. MetaEdit+ [Met15] was the Language Workbench (LW) [KLR96] applied to implement Canopus. The LW allows one to design the graphical elements that compose our DSL. Hence, the code generation module of MetaEdit+ makes it possible to transform the graphical representation in a textual representation. Thereby, to design our textual representation, we extend the Gherkin [WH12] language to include performance testing information. We chose to implement both graphical and textual languages, because even though graphical may be more rich visually, not all information can be represented. It important to highlight that the standard modeling language is graphical. The architecture of Canopus is composed of three parts: monitoring, scenario and scripting. Each part is represented by a metamodel, which in turn compounds a set of concepts of the performance domain.

Mechanisms to support the integration with third-party performance tools: Mechanisms to support the integration with third-party performance tools: the MetaEdit+ [Met15] has 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. Also, we were careful about the integration with third-party performance tools. 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 [Hew15]. Even so, a particular script generation for each technology as a new feature of Canopus can be also implemented. However, our proposal for model-based performance testing process supports the generation performance artifacts to specific technologies parsing the XML Canopus using PLeTsPerf [RBC+15], which in turn supports automatic generation of test scenarios and scripts to third-party performance tools.

Model-based performance testing process: Our DSL was designed to use integrated to an MBT approach. Thus, we propose a model-based performance testing process (see Section 5.2), which uses Canopus as a performance testing 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 generation of textual representation, generation of XML, and generation of third-party scripts. Hence, these activities produce and/or consume as input or as output testing artifacts generated in previous activities.

Industrial case study validation: We validated Canopus and its representativeness, intuitiveness, and usefulness in an industrial case study (Chapter 5). This case study applied the process mentioned in Section 5.2. Thus, an experience with real-world applications - Changepoint - within an IT corporation was reported (see Section 5.3). This case study provided evidence that Canopus is feasible in real and less controlled contexts. Moreover, an analysis (see Section 5.4) of a web-based survey to evaluate the graphical elements and their representativeness and intuitiveness to symbolize performance elements, was answered by fifteen performance test experts. The case study results also provided empirical evidence that the DSL proposed can be applied to larger and complex software projects, having an excellent analysis from the view point of the communication among stakeholders. In addiction to this, the performance team argued that Canopus could replace the textual specification used nowadays since the team evaluated that both languages, graphical and textual, represent their necessary specification to deal with other teams engaged in the software project. Furthermore, the main lessons we have learned from this industrial case study is summarized in Section 5.5.

Experimental evaluation of the proposed DSL: We designed and conducted a controlled empirical experiment to compare two approaches for modeling performance testing (Section 6.2). The experiment objective was to support our partner company in the decision process to replace UML by a DSL (Section 6.3). In Chapter 6, we validated Canopus report of an *in vitro* experiment, where the subjects analyze the design of annotated UML models and Canopus Performance models. This was done, 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 academic environments for modeling performance testing (Section 6.4). Thus, the experiment was executed with twenty-six subjects associated with two groups (one-half each): inexperienced and experienced. Moreover, the experiment results (Section 6.6) provide evidence of the benefits and drawbacks when using UML or DSL for modeling performance testing. Our findings indicate that, for performance modeling, the effort of using a DSL was lower than that of using UML. Our statistical analysis showed that the results were valid, *i.e.*, that for designing performance testing models, using our DSL is better than using UML.

Table 7.1 binds the stated achievements with three types of contributions: (i) theoretical definitions, (ii) experimental studies, and (iii) supporting tools. The table also presents the section in which the contribution was described and associated publication.

Table 7.1: Classification and structure of the achieved thesis contributions

Chapter	Description	Location	(i) Theoretical Definitions	(ii) Empirical Studies	(iii) Supporting Tools	Publication
4	Domain analysis for performance testing.	Section 4.2	✓			[BZR+14]
	Language requirements collected from the software engineers from TDL based on our domain analysis.	Section 4.3	✓			
	Design decisions for creating a DSL that supports the requirements colleted.	Section 4.4	✓			
	Proposal of the architecture of the DSL.	Section 4.5	✓			
	Proposal of the set of elements that composes the metamodels of the DSL.	Section 4.6			✓	
	Example of the feasiabity of using TPC-W to validate the applicability of the DSL.	Section 4.7		✓		
	Lessons learned to use the proposal DSL to evoluate de language.	Section 4.8		✓		
5	Model-based performance testing process.	Section 5.2	✓			[BRZ16]
	An exploratory study, the Changepoint case study.	Section 5.3		✓		
	Representativeness, intuitiveness, and useful analysis of the DSL.	Section 5.4		✓		
	Lessons learned from industrial experience.	Section 5.5		✓		
6	Instruments used during experiment training and execution.	Section 6.3		✓		[BZR16]
	Design and plan of the controlled empirical experiment.	Section 6.5		✓		
	Operation of the of the controlled empirical experiment	Section 6.5		✓		
	Effort, expressiveness, representativeness, intuitiveness, and easy-to-design analysis of the experiment results from the comparizon between UML and DSL aproaches.	Section 6.6		✓		

7.3 Limitations and Future Works

Despite this thesis having presented real contributions to the performance testing, model-based testing, and domain-specific language areas, we identified limitations of the thesis contributions that can be dealt with in the future. It is worth remembering that previous chapters already discussed specific limitations. Hence, we herein describe in this section 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 for performance testing 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 `Canopus Performance Monitoring` metamodel needs to add new properties to SUT, Load Generator, and Monitor elements - those represented by computers. For instance, besides IP address and hostname, it is interesting to maintain the basic information about its configuration, *e.g.* operating system (OS). 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, *i.e.*, if a new metric is created, this new feature must be added manually in Canopus to generate a new DSL version. Next, the `Canopus Performance 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. It would be interesting to create a mechanism by which to enable a relationship between `Canopus Performance Scenario` models, *i.e.*, a scenario model can decompose into another scenario model. Finally, during the case study with real-world applications shown in Chapter 5, and empirical experiment presented in Chapter 6, limitations on the `Canopus Performance Scripting` metamodel were identified. Hence, some graphical elements have not classified as intuitive. For instance, the Activity element has several forms and ways to draw, depending on the value of its properties. Further investigations into strategies for designing these graphical elements can shed some light on this topic.

Canopus is not exclusive to performance testing: 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 this scope. In future work, we may extend Canopus for other contexts or, who knows, to other testing paradigms, *e.g.* functional testing. For this reason, the name attributed for each metamodel takes the performance on its behalf. As a first interest, our studies with a TDL partner had the interest in investigating improvements for the performance testing process. Nowadays, the research migrated towards functional testing with the intention of the automating testing process.

Multiple industrial case studies: We conducted a single case study to find evidence of applicability our DSL. Hence, we understand that having applied Canopus addressing only one application, it is a threat to the conclusions of the generalization the viability of Canopus.

Thus, we plan to perform a multiple case study in a real context. Using multiple sources of evidence are essential tactic to collect data from several distinct software projects. However, this method is a challenge that makes the research becomes "hard" since we need to converge retrieved data in a triangulating fashion [Yin13]. Therefore, we will have software projects with different complexity and size of models for designing, as well as more subjects to interview or answer a survey to perform a qualitative analysis and then generalize these findings.

Replication of controlled empirical experiment: Despite the fact that we conducted a controlled empirical experiment to validate our results, we were aware that the scope of the number of experiment subjects that participated was limited, and opportunities to investigate better benefits were manifold. Thus, in this experiment, the most interesting dimensions were the effort (time spent/cost) 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 during experiment execution. This dimension characterizes the effectiveness of the performance models designed. The error rate is an important dimension to guarantee that the difficulty level of the tasks performed by experiment subject is independent of each approach. 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 techniques: As already stated, we performed an experiment to compare UML models and DSL models for modeling performance testing. We decided to make this comparison between two equivalent approaches to model performance testing based on the MBT paradigm. However, for most of the industry players, MBT is not yet a standard, for this reason, we believe that an empirical study comparing a Capture and Replay (CR) [EAXD$^+$10] based technique is necessary. Therefore, the empirical investigation between DSL-based and CR-based approaches also requires evidence. In the previous study, our research group performed a similar comparison. In this context, we conducted an experiment to compare a UML-based approach and a CR-based approach [RdOC$^+$15].

Performance testing body of knowledge: During the domain analysis of the performance testing, we investigated a broad scope of literature as well as practiced several software projects that applied performance testing. Besides, another recent study previously developed in our research group proposed an ontology for performance testing [FV14], mapping the main concepts, relationships, and constraints that represent the performance testing domain. Because of that, we are already sketching an initial performance testing body of knowledge. Our goals with this work are reaching the consensus through the body of knowledge and their set of core activities. Our aims are towards the evolution of performance testing professional status. This "guide" is a small alpha version of our body of knowledge that compose the core of performance testing activities, which are divided into their knowledge areas, as well as composed of their process groups. We intend to strive for this document to become a guide recognized and

adopted by the software industry. For this reason, we claim the participation and engagement of the industry and researchers interested in the performance testing area to write this guide with multi-hands to achieve such a consensus.

Human–Computer Interaction (HCI) evaluation of 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. In the meantime, there are other research fields in our Postgraduate Program in Computer Science. One of them investigates the HCI [Boo89] in software engineering. HCI studies the design and use of computer technology and its interface with users. Thus, HCI is an iterative cycle through analysis, design, implementation, and evaluation. Hence, usability engineering is applied based on human activity factors to deal with software projects. Jakob Nielsen proposed general principles for interaction design. These principles are called "heuristics" since they are wide general guidelines and not a particular guide of usability [NM90]. Some examples of these heuristics are as follows: consistency and standards, error prevention, flexibility and efficiency of use, help and documentation, and others. Therefore, we already began to investigate how to evaluate the usability heuristics for user interface design of Canopus.

7.4 Publications

During the development of this thesis we presented and discussed our research results in the following papers:

- Bernardino, M.; Zorzo, A.; Rodrigues, E. "Canopus: A Domain-Specific Language for Modeling Performance Testing", **forthcoming**. In: Proceedings of the 9^{th} IEEE International Conference on Software Testing, Verification and Validation (ICST'16), 2016, pp. 1–11.

- Bernardino, M.; Rodrigues, E.; Zorzo, A. "Performance Testing Modeling: an empirical evaluation of DSL and UML-based approaches", **forthcoming**. In: Proceeding of the 31^{st} ACM Symposium on Applied Computing (SAC-SVT'16), 2016, pp. 1–6.

- Bernardino, M.; Zorzo, A. F.; Rodrigues, E.; de Oliveira, F. M.; Saad, R. "A Domain-Specific Language for Modeling Performance Testing: Requirements Analysis and Design Decisions". In: Proceedings of the 9^{th} International Conference on Software Engineering Advances (ICSEA'14), 2014. pp. 609–614.

Also, we have a submission under review for a journal, as follows:

- Bernardino, M.; Rodrigues, E.; Zorzo, A. "A Systematic Mapping Study on Model-Based Testing: Tools and Models", **under review**, *IET Software*, 2016.

As we already stated, part of this research was performed in collaboration with a TDL of an IT company. Therefore, the research group collaborates to assist our partner as well as to produce research studies related to our initial proposal. Hence, we present a list of the other related publications that contributed indirectly to this thesis:

- Rodrigues, E.; Bernardino, M.; Costa, L.; Zorzo, A.; Oliveira, F. " PLeTsPerf - A Model-Based Performance Testing Tool". In: Proceedings of the 8^{th} IEEE International Conference on Software Testing, Verification and Validation (ICST'15), 2015, pp. 1–8.

- Rodrigues, E.M.; De Oliveira, F.; Bernardino, M.; Costa, L.T.; Zorzo, A.F.; Souza, S.R.S.; Saad, R. "An Empirical Comparison of Model-based and Capture and Replay Approaches for Performance Testing". *Empirical Software Engineering* (Online), v.20, n.6, pp. 1831–1860, 2015.

- Rodrigues, E. M.; Saad, R. S.; Oliveira, F. M.; Costa, L. T.; Bernardino, M.; Zorzo, A. F. "Evaluating Capture and Replay and Model-based Performance Testing Tools: An Empirical Comparison". In: Proceedings of the 8^{th} ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'14), 2014. pp. 9:1–9:8.

- Costa, L.T.; Zorzo, A.F.; Rodrigues, E.M.; Bernardino, M.; Oliveira, F.M. "Structural Test Case Generation Based on System Models". In: Proceedings of the 9^{th} International Conference on Software Engineering Advances (ICSEA'14), 2014. pp. 276–281.

- Guarianti, P.; Bernardino, M.; Zorzo, A.F.; Oliveira, F.M. "Uma Abordagem de Análise do Tempo de Resposta para Teste de Desempenho em Aplicações Web". In: Anais do XV Workshop de Testes e Tolerância a Falhas (WTF'14), 2014. p. 17–30.

- Endo, A.T.; Bernardino, M.; Rodrigues, E.M.; Simao, A.; Oliveira, F.M.; Zorzo, A.F.; Saad, R. "An Industrial Experience on using Models to Test Web Service-Oriented Applications". In: Proceedings of the 15^{th} International Conference on Information Integration and Web-based Applications & Services (iiWAS'13), 2013. pp. 240–249.

- Costa, L.T.; Czekster, R.M.; Oliveira, F.M.; Rodrigues, E.M.; Silveira, M.B. ; Zorzo, A.F. "Generating Performance Test Scripts and Scenarios Based on Abstract Intermediate Models". In: Proceedings of the 24^{th} International Conference on Software Engineering and Knowledge Engineering (SEKE'12), 2012. v.1. pp. 112–117.

REFERENCES

[AATP12] Abbors, F.; Ahmad, T.; Truscan, D.; Porres, I. "MBPeT - A Model-Based Performance Testing Tool". In: 4^{th} International Conference on Advances in System Testing and Validation Lifecycle, 2012, pp. 1–8.

[AFG+10] Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A. D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; Zaharia, M. "A View of Cloud Computing", *Communication ACM*, vol. 53–4, apr 2010, pp. 50–58.

[ALRL04] Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C. "Basic Concepts and Taxonomy of Dependable and Secure Computing", *IEEE Transaction on Dependable Secure Computing*, vol. 1–1, 2004, pp. 11–33.

[Ast15] Astah. "Astah Professional". Available in: http://astah.net/, Dec 2015.

[AT10] Abbors, F.; Truscan, D. "Approaching Performance Testing from a Model-Based Testing Perspective". In: 2^{nd} International Conference on Advances in System Testing and Validation Lifecycle, 2010, pp. 125–128.

[Bac78] Backus, J. "Can Programming Be Liberated from the Von Neumann Style?: A Functional Style and Its Algebra of Programs", *Communications of the ACM*, vol. 21–8, aug 1978, pp. 613–641.

[Bar04] Barber, S. "Creating Effective Load Models for Performance Testing with Incomplete Empirical Data". In: 6^{th} IEEE International Workshop on Web Site Evolution, 2004, pp. 51–59.

[Bar15] Barber, S. "User Community Modeling Language (UCML) for performance test workloads". Available in: http://www.ibm.com/developerworks/rational/library/5219.html, Sep 2015.

[BCM07] BCM Software. "Performance Benchmarking Kit Using Incident Management with SilkPerformer", Technical Report, BMC Software, 2007, 66p.

[BCW12] Brambilla, M.; Cabot, J.; Wimmer, M. "Model-Driven Software Engineering in Practice". San Rafael, CA, USA: Morgan & Claypool Publishers, 2012, 1^{st} ed., 182p.

[BDG+07] Baker, P.; Dai, Z. R.; Grabowski, J.; Haugen, O.; Schieferdecker, I.; Williams, C. "Model-Driven Testing: Using the UML Testing Profile". Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007, 184p.

[BDH06] Beyer, M.; Dulz, W.; Hielscher, K.-S. "Performance Issues in Statistical Testing". In: 13^{th} GI/ITG Conference Measuring, Modelling and Evaluation of Computer and Communication Systems, 2006, pp. 1–17.

[BDMIS04] Balsamo, S.; Di Marco, A.; Inverardi, P.; Simeoni, M. "Model-based performance prediction in software development: A survey", *IEEE Transactions on Software Engineering*, vol. 30–5, may 2004, pp. 295–310.

[Bea78] Beaudry, M. D. "Performance-Related Reliability Measures for Computing Systems", *IEEE Transactions on Computers*, vol. C-27–6, jun 1978, pp. 540–547.

[Bea15] Bear, S. "Software LoadUI". Available in: http://www.loadui.org, Jan 2015.

[Bec02] Beck. "Test Driven Development: By Example". Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002, 240p.

[Ber07] Bertolino, A. "Software Testing Research: Achievements, Challenges, Dreams". In: Future of Software Engineering, 2007, pp. 85–103.

[Bie06] Bierhoff, Kevin; Liongosari, Edy S.; Swaminathan, Kishore S. "Incremental Development of a Domain-Specific Language That Supports Multiple Application Styles". In: OOPSLA Workshop on Domain-Specific Modeling, 2006, pp. 79–86.

[Boo89] Booth, P. A. "An Introduction to Human-Computer Interaction". Hove, GB, Hillsdale, NJ: Lawrence Erlbaum Associates, 1989, 268p.

[BRZ16] Bernardino, M.; Rodrigues, E.; Zorzo, A. "Performance Testing Modeling: an empirical evaluation of DSL and UML-based approaches *(forthcoming)*". In: 31^{st} ACM Symposium on Applied Computing, 2016, pp. 1–6.

[BvHH+04] Bechhofer, S.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; Stein, L. A. "OWL Web Ontology Language Reference", Technical Report, W3C, 2004.

[BZGL07] Bui, N.; Zhu, L.; Gorton, I.; Liu, Y. "Benchmark Generation Using Domain Specific Modeling". In: 18^{th} Australian Software Engineering Conference, 2007, pp. 169–180.

[BZR+14] Bernardino, M.; Zorzo, A. F.; Rodrigues, E.; de Oliveira, F. M.; Saad, R. "A Domain-Specific Language for Modeling Performance Testing: Requirements Analysis and Design Decisions". In: 9^{th} International Conference on Software Engineering Advances, 2014, pp. 609–614.

[BZR16] Bernardino, M.; Zorzo, A.; Rodrigues, E. "Canopus: A Domain-Specific Language for Modeling Performance Testing *(forthcoming)*". In: 9^{th} International Conference on Software Testing, Verification and Validation, 2016, pp. 1–8.

[CC79] Cook, T. D.; Campbell, D. T. "Quasi-Experimentation: Design and Analysis Issues for Field Settings". Boston, MA, USA: Houghton Mifflin, 1979, 420p.

[CCO+12] Costa, L. T.; Czekster, R.; Oliveira, F. M.; Rodrigues, E. M.; Silveira, M. B.; Zorzo, A. F. "Generating Performance Test Scripts and Scenarios Based on Abstract Intermediate Models." In: 24^{th} International Conference on Software Engineering and Knowledge Engineering, 2012, pp. 112–117.

[CDD+08] Chadwick, D.; Davis, C.; Dunn, M.; Jessee, E.; Kofaldt, A.; Mooney, K.; Nicolas, R.; Patel, A.; Reinstrom, J.; Siefkes, K.; Silva, P.; Ulrich, S.; Yeung, W. "Using Rational Performance Tester Version 7". Riverton, NJ, USA: IBM Redbooks, 2008, 572p.

[Cha94] Chang, C.-S. "Stability, Queue Length, and Delay of Deterministic and Stochastic Queueing Networks", *IEEE Transactions on Automatic Control*, vol. 39–5, may 1994, pp. 913–931.

[Cha15] Changepoint. "Changepoint". Available in: http://www.changepoint.com, Dec 2015.

[Cho78] Chow, T. S. "Testing Software Design Modeled by Finite-State Machines", *IEEE Transaction on Software Engineering*, vol. 4–3, may 1978, pp. 178–187.

[CNN01] Clements, P.; Northrop, L.; Northrop, L. M. "Software Product Lines: practices and patterns". Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001, 608p.

[CPL09] Chung, L.; Prado Leite, J. C. "Conceptual Modeling: Foundations and Applications". In: *On Non-Functional Requirements in Software Engineering*, Borgida, A. T.; Chaudhri, V. K.; Giorgini, P.; Yu, E. S. (Editors), Berlin, Heidelberg: Springer-Verlag, 2009, vol. 5600, pp. 363–379.

[CR09] Casey, V.; Richardson, I. "Implementation of Global Software Development: A Structured Approach", *Software Process*, vol. 14–5, sep 2009, pp. 247–262.

[dNVN+12] do Nascimento, L. M.; Viana, D. L.; Neto, P. A. M. S.; Martins, D. A. O.; Garcia, V. C.; Meira, S. R. L. "Systematic Mapping Study on Domain-Specific Languages". In: 7^{th} International Conference on Software Engineering Advances, 2012, pp. 179–187.

[dOMVR07] de Oliveira, F. M.; Menna, R. d. S.; Vieira, H. V.; Ruiz, D. D. A. "Performance Testing from UML Models with Resource Descriptions". In: 1^{st} Brazilian Workshop on Systematic and Automated Software Testing, 2007, pp. 47–54.

[dSdMRZ+11] da Silveira, M. B.; de M. Rodrigues, E.; Zorzo, A. F.; Costa, L. T.; Vieira, H. V.; de Oliveira, F. M. "Generation of Scripts for Performance Testing Based on UML Models". In: 23^{rd} International Conference on Software Engineering and Knowledge Engineering, 2011, pp. 258–263.

[DTA+08] Demathieu, S.; Thomas, F.; Andre, C.; Gerard, S.; Terrier, F. "First Experiments Using the UML Profile for MARTE". In: 11^{th} IEEE International Symposium on Object Oriented Real-Time Distributed Computing, 2008, pp. 50–57.

[EAXD+10] El Ariss, O.; Xu, D.; Dandey, S.; Vender, B.; McClean, P.; Slator, B. "A Systematic Capture and Replay Strategy for Testing Complex GUI Based Java Applications". In: 7^{th} International Conference on Information Technology: New Generations, 2010, pp. 1038–1043.

[Eck95] Eckerson, W. W. "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications.", *Open Information Systems*, vol. 10–1, 1995.

[Edm73] Edmonds, Jack; Johnson, E. L. "Matching, Euler tours and the Chinese postman", *Mathematical Programming*, vol. 5–1, 1973, pp. 88–124.

[EMF15] EMF. "Eclipse Modeling Framework". Available in: http://www.eclipse.org/modeling/emf/, Dec 2015.

[ESV+13] Erdweg, S.; Storm, T.; Völter, M.; Boersma, M.; Bosman, R.; Cook, W.; Gerritsen, A.; Hulshout, A.; Kelly, S.; Loh, A.; Konat, G. D.; Molina, P.; Palatnik, M.; Pohjonen, R.; Schindler, E.; Schindler, K.; Solmi, R.; Vergu, V.; Visser, E.; Vlist, K.; Wachsmuth, G.; Woning, J. "The State of the Art in Language Workbenches". In: *Software Language Engineering*, Erwig, M.; Paige, R.; Wyk, E. (Editors), Springer International Publishing, 2013, vol. 8225, pp. 197–217.

[FK05] Frakes, W. B.; Kang, K. "Software Reuse Research: status and future", *IEEE Transactions on Software Engineering*, vol. 31–7, jul 2005, pp. 529–536.

[Fow09] Fowler, M. "A Pedagogical Framework for Domain-Specific Languages", *IEEE Software*, vol. 26–4, july-aug 2009, pp. 13–14.

[Fow10] Fowler, M. "Domain Specific Languages". Boston, MA, USA: Addison-Wesley Professional, 2010, 1^{st} ed., 640p.

[FV14] Freitas, A.; Vieira, R. "An Ontology for Guiding Performance Testing". In: IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies, 2014, pp. 400–407.

[Gat15] Gatling. "Gatling Stress Tool". Available in: http://gatling.io/, Dec 2015.

[GFC+08] Gray, J.; Fisher, K.; Consel, C.; Karsai, G.; Mernik, M.; Tolvanen, J.-P. "DSLs: the Good, the Bad, and the Ugly". In: Companion to the 23^{rd} ACM Special Interest Group on Programming Languages Conference on Object-Oriented Programming Systems Languages and Applications, 2008, pp. 791–794.

[Gho11] Ghosh, D. "DSL for the Uninitiated", *Queue*, vol. 9–6, Jun 2011, pp. 10:10–10:21.

[Gil62] Gill, A. "Introduction to the Theory of Finite State Machines". New York, NY, USA: McGraw-Hill, 1962, 207p.

[GK09] Gherbi, A.; Khendek, F. "From UML/SPT Models to Schedulability Analysis: Approach and a Prototype Implementation Using ATL", *Automated Software Engineering*, vol. 16–3-4, Dec 2009, pp. 387–414.

[GME15] GME. "Generic Modeling Environment". Available in: http://www.isis.vanderbilt.edu/projects/gme, Dec 2015.

[GMP15] GMP. "Graphical Modeling Project". Available in: http://www.eclipse.org/modeling/gmp/, Dec 2015.

[GZ15] Gómez, P.; Zadrozny, P. "Software Grinder". Available in: http://grinder.sourceforge.net, Jan 2015.

[Hew15] Hewlett Packard HP. "Software HP LoadRunner". Available in: http://goo.gl/JU2R5d, Dec 2015.

[HGB08] Hasling, B.; Goetz, H.; Beetz, K. "Model Based Testing of System Requirements using UML Use Case Models". In: 1^{st} International Conference on Software Testing, Verification, and Validation, 2008, pp. 367–376.

[HT06] Hailpern, B.; Tarr, P. "Model-Driven Development: The Good, the Bad, and the Ugly", *IBM Systems Journal*, vol. 45–3, Jul 2006, pp. 451–461.

[IEE90] "IEEE Standard Glossary of Software Engineering Terminology", *IEEE Std 610.12-1990*, Dec 1990, pp. 1–84.

[IEE08] "IEEE Standard for Software and System Test Documentation", *IEEE Std 829-2008*, July 2008, pp. 1–150.

[Jet15] JetBrains. "Meta Programming System". Available in: http://www.jetbrains.com/mps, Dec 2015.

[JLH+10] Jing, Y.; Lan, Z.; Hongyuan, W.; Yuqiang, S.; Guizhen, C. "JMeter-Based Aging Simulation of Computing System". In: International Conference on Computer, Mechatronics, Control and Electronic Engineering, 2010, pp. 282–285.

[KBJV06] Kurtev, I.; Bézivin, J.; Jouault, F.; Valduriez, P. "Model-Based DSL Frameworks". In: Companion to the 21^{st} Special Interest Group on Programming Languages Symposium on Object-oriented programming systems, languages, and applications, 2006, pp. 602–616.

[KCH+90] Kang, K. C.; Cohen, S. G.; Hess, J. A.; Novak, W. E.; Peterson, A. S. "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Technical Report CMU/SEI-90-TR-21, Carnegie-Mellon University Software Engineering Institute, 1990, 161p.

[KLR96] Kelly, S.; Lyytinen, K.; Rossi, M. "MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment". In: 8^{th} International Conference on Advances Information System Engineering, 1996, pp. 1–21.

[KRM06] Krishnamurthy, D.; Rolia, J. A.; Majumdar, S. "A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems", *IEEE Transaction Software Engineering*, vol. 32–11, nov 2006, pp. 868–882.

[Kru92] Krueger, C. W. "Software Reuse", *ACM Computing Surveys*, vol. 24–2, jun 1992, pp. 131–183.

[KSF10] Krishnamurthy, D.; Shams, M.; Far, B. H. "A Model-Based Performance Testing Toolset for Web Applications", *Engineering Letters*, vol. 18–2, may 2010, pp. 92–106.

[KT07] Kelly, S.; Tolvanen, J.-P. "Domain-Specific Modeling: Enabling Full Code Generation". New York, NY, USA: John Wiley & Sons, 2007, 444p.

[LBN02] Lee, J.; Ben-Natan, R. "Integrating Service Level Agreements: Optimizing Your OSS for SLA Delivery". New York, NY, USA: John Wiley & Sons, 2002, 464p.

[Lik32] Likert, R. "A Technique for the Measurement of Attitudes", *Archives of Psychology*, vol. 140–55, 1932.

[LJJ07] Langlois, B.; Jitia, C.; Jouenne, E. "DSL Classification". In: 7th OOPSLA Workshop on Domain-Specific Modeling, 2007, pp. 1–11.

[LMdG+09] Lamancha, B. P.; Mateo, P. R.; de Guzmán, I. R.; Usaola, M. P.; Velthius, M. P. "Automated Model-Based Testing Using the UML Testing Profile and QVT". In: 6^{th} International Workshop on Model-Driven Engineering, Verification and Validation, 2009, pp. 6:1–6:10.

[LW08] Lutteroth, C.; Weber, G. "Modeling a Realistic Workload for Performance Testing". In: 12^{th} International IEEE Enterprise Distributed Object Computing Conference, 2008, pp. 149–158.

[LY96] Lee, D.; Yannakakis, M. "Principles and Methods of Testing Finite State Machines-A Survey", *Proceedings of the IEEE*, vol. 84–8, 1996, pp. 1090–1123.

[MAFM99] Menascé, D. A.; Almeida, V. A. F.; Fonseca, R.; Mendes, M. A. "A Methodology for Workload Characterization of E-commerce Sites". In: 1^{st} ACM Conference on Electronic Commerce, 1999, pp. 119–128.

[MDA04] Menascé, D. A.; Dowdy, L. W.; Almeida, V. A. F. "Performance by Design: Computer Capacity Planning By Example". Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004, 480p.

[MDN09] Mohagheghi, P.; Dehlen, V.; Neple, T. "Definitions and approaches to model quality in model-based software development - a review of literature", *Information and Software Technology*, vol. 51, dec 2009, pp. 1646–1669.

[Men02] Menascé, D. "TPC-W: a Benchmark for e-Commerce", *IEEE Internet Computing*, vol. 6–3, 2002, pp. 83–87.

[Met15] MetaCase. "MetaEdit+". Available in: http://www.metacase.com/mep/, Dec 2015.

[MFB+07] Meier, J.; Farre, C.; Bansode, P.; Barber, S.; Rea, D. "Performance Testing Guidance for Web Applications: Patterns & Practices". Redmond, WA, USA: Microsoft Press, 2007, 221p.

[MFV06] Moreno, N.; Fraternalli, P.; Vallecillo, A. "A UML 2.0 Profile for WebML Modeling". In: 6^{th} International Conference on Web Engineering, 2006, pp. 11–20.

[MHS05] Mernik, M.; Heering, J.; Sloane, A. M. "When and How to Develop Domain-Specific Languages", *ACM Computing Surveys*, vol. 37–4, Dec 2005, pp. 316–344.

[Mic15a] Microsoft. "DSL Tools web site". Available in: http://msdn.microsoft.com/en-us/library/bb126259.aspx, Dec 2015.

[Mic15b] Microsoft. "Visual Studio". Available in: http://www.visualstudio.com, Dec 2015.

[MLB+11] Marston, S.; Li, Z.; Bandyopadhyay, S.; Zhang, J.; Ghalsasi, A. "Cloud Computing: The Business Perspective", *Decision Support Systems*, vol. 51–1, 2011, pp. 176–189.

[Mol82] Molloy, M. K. "Performance Analysis Using Stochastic Petri Nets", *IEEE Transactions on Computers*, vol. 31–9, 1982, pp. 913–917.

[Mol09] Molyneaux, I. "The Art of Application Performance Testing: Help for Programmers and Quality Assurance". Sebastopol, CA, USA: O'Reilly Media, 2009, 1^{st} ed., 158p.

[Moo15] Moodle Pty Ltd. "Software Moodle". Available in: http://moodle.org, Jan 2015.

[MV00] Menascé, D. A.; Virgilio, A. F. A. "Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning". Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000, 1^{st} ed., 462p.

[Neo15] Neotys. "Neo Load". Available in: http://www.neotys.com/neoload, Jan 2015.

[NM90] Nielsen, J.; Molich, R. "Heuristic Evaluation of User Interfaces". In: ACM Special Interest Group on Computer-Human Interaction Conference on Human Factors in Computing Systems, 1990, pp. 249–256.

[Oat06] Oates, B. J. "Researching Information Systems and Computing". London, UK: SAGE Publications, 2006, 360p.

[OMG05] OMG. "UML Profile for Schedulability, Performance, and Time Specification - OMG Adopted Specification Version 1.1, formal/05-01-02". Available in: http://www.omg.org/spec/SPTP/1.1/PDF, Jan 2005.

[OMG09] OMG. "UML Profile for Modeling and Analysis of Real-Time and Embedded Systems - MARTE specification v.1.0 (2009-11-02)". Available in: http://www.omgmarte.org, Nov 2009.

[OMG15] OMG. "Unified Modeling Language". Available in: http://www.uml.org/, Dec 2015.

[PA91] Plateau, B.; Atif, K. "Stochastic Automata Network of Modeling Parallel Systems", *IEEE Transactions on Software Engineering*, vol. 17–10, oct 1991, pp. 1093–1108.

[PBvdL05] Pohl, K.; Böckle, G.; van der Linden, F. J. "Software Product Line Engineering: Foundations, Principles and Techniques". Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005, 467p.

[PDA91] Prieto-Diaz, R.; Arango, G. "Domain Analysis and Software Systems Modeling". Los Alamitos, CA, USA: IEEE Computer Society Press, 1991, 312p.

[PG11] Pozin, B. A.; Galakhov, I. V. "Models in Performance Testing", *Programming and Computer Software*, vol. 37–1, jan 2011, pp. 15–25.

[Por15] Portal Action. "System Action Statistical Package". Available in: http://www.portalaction.com.br, Dec 2015.

[PW04] Petriu, D. B.; Woodside, M. "A Metamodel for Generating Performance Models from UML Designs". In: *UML 2004 - The Unified Modelling Language*, Baar, T.; Strohmeier, A.; Moreira, A.; Mellor, S. J. (Editors), Springer Berlin / Heidelberg, 2004, vol. 3273, pp. 41–53.

[RBC+15] Rodrigues, E.; Bernardino, M.; Costa, L.; Zorzo, A.; Oliveira, F. "PLeTsPerf - A Model-Based Performance Testing Tool". In: 8^{th} International Conference on Software Testing, Verification and Validation, 2015, pp. 1–8.

[RdOC+15] Rodrigues, E. M.; de Oliveira, F. M.; Costa, L. T.; Bernardino, M.; Zorzo, A. F.; Souza, S. d. R. S.; Saad, R. "An Empirical Comparison of Model-Based and Capture and Replay Approaches for Performance Testing", *Empirical Software Engineering*, vol. 20–6, 2015, pp. 1831–1860.

[RSO+14] Rodrigues, E. M.; Saad, R. S.; Oliveira, F. M.; Costa, L. T.; Bernardino, M.; Zorzo, A. F. "Evaluating Capture and Replay and Model-based Performance Testing Tools:

An Empirical Comparison". In: 8^{th} ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2014, pp. 9:1–9:8.

[RSSP04] Ruffo, G.; Schifanella, R.; Sereno, M.; Politi, R. "WALTy: A User Behavior Tailored Tool for Evaluating Web Application Performance". In: 3^{rd} IEEE International Symposium on Network Computing and Applications, 2004, pp. 77–86.

[RVZG10] Rodrigues, E. M.; Viccari, L. D.; Zorzo, A. F.; Gimenes, I. M. "PLeTs Tool - Test Automation using Software Product Lines and Model Based Testing". In: 22th International Conference on Software Engineering and Knowledge Engineering, 2010, pp. 483–488.

[SA98] Simos, M.; Anthony, J. "Weaving the Model Web: a Multi-Modeling Approach to Concepts and Features in Domain Engineering". In: 5^{th} International Conference on Software Reuse, 1998, pp. 94–102.

[SBF14] Society, I. C.; Bourque, P.; Fairley, R. E. "Guide to the Software Engineering Body of Knowledge (SWEBOK): Version 3.0". Los Alamitos, CA, USA: IEEE Computer Society Press, 2014, 3rd ed., 346p.

[Sch06] Schmidt, D. C. "Guest Editor's Introduction: Model-Driven Engineering", *Computer*, vol. 39–2, feb 2006, pp. 25–31.

[SCL91] Sampieri, R. H.; Collado, C. F.; Lucio, P. B. "Metodologia de la Investigación". Ciudad de México, México: McGraw Hill, 1991, 200p.

[Sel07] Selic, B. "A Systematic Approach to Domain-Specific Language Design Using UML". In: 10^{th} IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2007, pp. 2–9.

[Ser95] Serain, D. "Client/server: Why? What? How?" In: International Seminar on Client/Server Computing. Seminar Proceedings (IEE Digest No. 1995/184), 1995, pp. 1–11.

[SKF06] Shams, M.; Krishnamurthy, D.; Far, B. "A Model-Based Approach for Testing the Performance of Web Applications". In: 3^{rd} International Workshop on Software Quality Assurance, 2006, pp. 54–61.

[Smi02] Smith, C. U. "Software Performance Engineering". New York, NY, USA: John Wiley & Sons, 2002, 570p.

[Sof10] Software Engineering Institute (SEI). "Software Product Lines (SPL)". Available in: http://www.sei.cmu.edu/productlines/, sep 2010.

[Spa12] Spafford, Kyle L. and Vetter, Jeffrey S. "Aspen: A Domain Specific Language for Performance Modeling". In: International Conference on High Performance Computing, Networking, Storage and Analysis, 2012, pp. 84:1–84:11.

[The15] The Eclipse Fundation. "Xtext - Language Development Framework". Available in: http://www.eclipse.org/Xtext/, Dec 2015.

[TMG09] Tairas, R.; Mernik, M.; Gray, J. "Models in Software Engineering". , Chaudron, M. R. (Editor), Berlin, Germany: Springer–Verlag, 2009, chap. Using Ontologies in the Domain Analysis of Domain-Specific Languages, pp. 332–342.

[TTC95] Taylor, R. N.; Tracz, W.; Coglianese, L. "Software Development Using Domain-specific Software Architectures: CDRl A011—a Curriculum Module in the SEI Style", *ACM Special Interest Group on Software Enginnering Notes*, vol. 20–5, dec 1995, pp. 27–38.

[TWESAO10] Traore, I.; Woungang, I.; El Sayed Ahmed, A.; Obaidat, M. "UML-based Performance Modeling of Distributed Software Systems". In: International Symposium on Performance Evaluation of Computer and Telecommunication Systems, 2010, pp. 119–126.

[UL06] Utting, M.; Legeard, B. "Practical Model-Based Testing: A Tools Approach". San Francisco, CA, USA: Morgan Kaufmann, 2006, 456p.

[vDKV00] van Deursen, A.; Klint, P.; Visser, J. "Domain-Specific Languages: An Annotated Bibliography", *ACM Special Interest Group on Programming Languages Notices*, vol. 35, Jun 2000, pp. 26–36.

[Vou08] Vouk, M. A. "Cloud Computing: Issues, Research and Implementations". In: 30^{th} International Conference on Information Technology Interfaces, 2008, pp. 31–40.

[WB10] Wu, L.; Buyya, R. "Service Level Agreement (SLA) in Utility Computing Systems", *Computing Research Repository (CoRR)*, vol. abs/1010.2881, Dec 2010.

[WFP07] Woodside, M.; Franks, G.; Petriu, D. C. "The Future of Software Performance Engineering". In: Future of Software Engineering, 2007, pp. 171–187.

[WH12] Wynne, M.; Hellesøy, A. "The Cucumber Book: Behaviour-Driven Development for Testers and Developers". The Pragmatic Bookshelf, 2012, 336p.

[WL99] Weiss, D. M.; Lai, C. T. R. "Software Product-Line Engineering: a family-based software development process". Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999, 426p.

[WRH+12] Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B. "Experimentation in Software Engineering". Berlin, Germany: Springer–Verlag, 2012, 1^{st} ed., 236p.

[WSPS09] Walter, T.; Silva Parreiras, F.; Staab, S. "OntoDSL: An Ontology-Based Framework for Domain-Specific Languages". In: 12^{th} International Conference on Model Driven Engineering Languages and Systems, 2009, pp. 408–422.

[YHL+08] Yang, Y.; He, M.; Li, M.; Wang, Q.; Boehm, B. "Phase Distribution of Software Development Effort". In: 2^{nd} ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2008, pp. 61–69.

[Yin13] Yin, R. "Case Study Research: Design and Methods". London, UK: SAGE Publications, 2013, 5^{th} ed., 312p.

[ZG07] Zhu, L.; Gorton, I. "UML Profiles for Design Decisions and Non-Functional Requirements". In: 2^{nd} Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent, 2007, pp. 8–8.

APPENDIX A – CANOPUS PERFORMANCE SPECIFICATION

A.1 Canopus Performance Monitoring Specification

Object: `SUT`
Description: System Under Test;
Properties:

Hostname (String): A label assigned to a computer;
IP (String): IP address;
Type ((String:Fixed List): Type of host, *e.g.* DesktopApp, Database, WebApp, WebService;
Hardware (String:Fixed List): Physical Machine (PM), Virtual Machine (VM), Cloud Service (CS);
Metrics (CanopusPerformanceMetric metamodel): Associated with `Canopus Performance Metric` model (see Section A.1.1). A monitor icon appears on the top-left side when there is an associated metric model.

WebApp (PM)
SUT Example
192.168.1.2

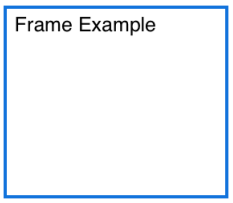
Object: `Load Generator`
Description: Load generator workload machine;
Properties:

Hostname (String): A label assigned to a computer;
IP (String): IP address;
Monitor (Boolean): Defines if the LG is also the monitoring machine, *i.e.*, the same machine exercises a double function. A magnifier icon appears on the top-left side of monitor when this property is set to true.;
Hardware (String:Fixed List): Physical Machine (PM), Virtual Machine (VM), Cloud Service (CS);
Metrics (Canopus Performance Metric metamodel): Associated with `Canopus Performance Metric` model (see Section A.1.1). A monitor icon appears on the top-left side when there is an associated metric model.

Load Generator (VM)
LG Example
192.168.1.3

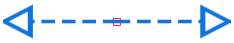
Object: `Monitor`
Description: Machine responsible for monitoring the performance metrics of the SUT. This object is optional, since the Load Generator object, besides generating workload for virtual users, can also play the role of monitoring;
Properties:
Hostname (String): A label assigned to a computer;
IP (String): IP address;
Hardware (String:Fixed List): Physical Machine (PM), Virtual Machine (VM), Cloud Service (CS);

Object: `Frame`
Description: Used to visually group other objects that compose the `Canopus Performance Monitoring` model. On the top-left side of frame the project name is rendered;
Properties: There are none.



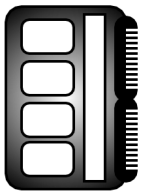
Relashionship: `Flow`
Description: Binds a Load Generator object or a Monitor object on a SUT (target of arrow) object, or binds a Monitor object on a Load Generator (target of arrow) object.
Properties: There are none.

Relashionship: `Association`
Description: Binds a SUT on another SUT. For instance, to demonstrate the communication networking between a Web application server and a database server.
Properties: There are none.



A.1.1 Canopus Performance Metric Specification

Object: `Memory`
Description: Represents the Memory metric that will be measured;
Properties:
Name (String): Metric name;



Object: `Memory Counter`
Description: A shortage of RAM is often evident indirectly as a disk performance problem, when excessive paging to disk consumes too much of the available disk bandwidth. Consequently, paging rates to disk are an important memory performance indicator. When observing a shortage of available RAM, it is often important to determine how the allocated physical memory is being used and count resident pages of a problematic process known as its

working set.

Properties:

Counter (String:Fixed List):

Available MBytes Counter: Indicates the amount of physical memory available to processes running on the computer;

Pages/Sec Counter: Indicates the number of resident pages of each process;

Page Reads/Sec Counter: Indicates the rate at which pages are read from or written to disk to resolve hard page faults;

Working Set Counter: Indicates that the working set of the process is too large for the physical memory and that it is paging to disk;

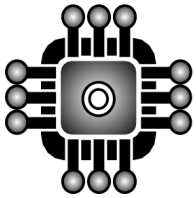
Pool Nonpaged Bytes Counter: Indicates the size of an area of system memory (physical memory used by the operating system) for objects that cannot be written to disk, but must remain in physical memory as long as they are allocated;

Paged Pool Bytes Counter: Indicates memory leaks;

Paged Pool Failures Counter: Indicates the number of times allocations from the paged pool have failed;

Cache Bytes Counter: Indicates the size of the static files cache.

Object: `Processor`

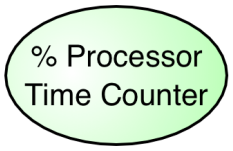


Description: Represents the processor (CPU) metric that will be measured;

Properties:

Name (String): Metric name;

Object: `Processor Counter`



Description: Program execution threads consume processor (CPU) resources. These threads can be part of user-mode processes or the operating system kernel. Available performance counters measure how much CPU processing time threads and other executable units of work consume. These processor utilization measurements allow to determine which applications are responsible for CPU consumption.

Properties:

Counter (String:Fixed List):

% Processor Time Counter: Indicates the percentage of elapsed time that the processor spends to execute a non-idle thread;

% Privileged Time Counter: Indicates the percentage of elapsed time that the process threads spent executing code in privileged mode;

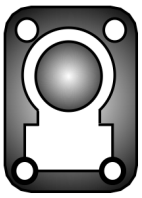
% Interrupt Time Counter: Indicates the time the processor spends receiving and servicing hardware interrupts during sample intervals;

Processor Queue Length Counter: Indicates the number of threads in the processor queue;

Context Switches Counter: Indicates the combined rate at which all processors on the computer are switched from one thread to another;

System Up Time Counter: Indicates the indicator of overall system availability;

Object: `Disk`

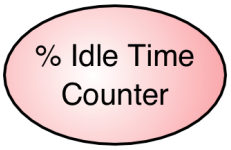


Description: Represents the Disk (I/O) metric that will be measured. Disk I/O latency can be defined as a measurement of the time delay from the time a disk I/O request is created, until the time the disk I/O request is completed.

Properties:

Name (String): Metric name;

Object: `Disk (I/O) Counter`



Description: Through the I/O Manager stack, an operation system maintains physical and logical disk operations. A logical disk represents a single file system with a unique drive letter. A physical disk is the internal representation of specific storage device - be it SCSI, RAID, SATA, or other technology. When using complex storage systems such as array controllers or RAID, the underlying physical disk hardware characteristics are not directly visible to the operating system. These characteristics - namely, the number of disks, the speed of the disks, their seek time, rotational speed, and bit density as well as some optimization features such as on-board memory buffers - can have a major impact on performance. Advance features like memory buffers and command-queueing can boost the performance by 25–50 percent. It is important to be proactive about disk performance because it tends to degrade rapidly, particularly when disk-paging activity occurs.

Properties:

Counter (String:Fixed List):

Avg. Disk secs/transfer Counter: Indicates physical disk potential bottleneck;

% Idle Time Counter: Indicates physical disk utilization;

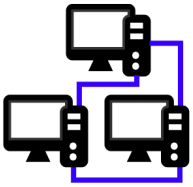
Disk Transfers/sec Counter: Indicates whether physical disk is a potential bottleneck;

Avg. Disk Queue Length Counter: Indicates, although in conjunction with other counters, a potential bottleneck of a disk;

Split IO/sec Counter: Indicates possible defragmentation;

Free Megabytes Counter: Indicates logical disk space usage.

Object: `Network`



Description: Represents the Network metric that will be measured.

Properties:

Name (String): Metric name;

Object: `Network Counter`

Description: Networking performance has become ever more important today with the proliferation of distributed and cloud applications. However, some operating systems usually provide limited statistics on various levels: At the lowest level hardware interface, and at higher level of network protocol such as TCP/IP. Network interface

statistics are gathered by software embedded in the network interface driver layer. This software counts the number of packets that are sent and received. Networking bottlenecks are tricky to catch and analyze. Packet rates, collision rates and error rates do not always point to the cause of the problem.

Properties:

Counter (String:Fixed List):

Bytes Total/Sec: Indicates total throughput;

Server Bytes Total/Sec: Indicates overall server utilization in terms of network;

Datagrams/Sec: Indicates IP protocol load;

Connections Established: Indicates TCP protocol connection success rate;

Segments Received/Sec: Indicates number of TCP data segments received

% Interrupt Time: Indicates the time the processor spends on hardware devices interrupts, such as network card.

Object: `Web Resources`

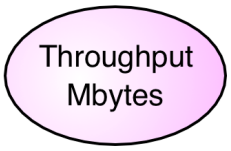
Description: Represents the WebResources metric that will be measured.

Properties:

Name (String): Metric name;

Object: `Web Resources Counter`

Description: These are vital counters for assessment of application ability to sustain the simulated workload.



Properties:

Counter (String:Fixed List):

Throughput Mbytes: Shows the amount of throughput on the server during each second of the load test scenario run. Throughput measures the actual rate at which work requests are completed. Throughput is measured in bytes or megabytes and represents the amount of data that the Vusers received from the server at any given second;

Hits per second: Shows the number of requests per second;

HTTP responses per second: shows the number of HTTP status codes returned from the Web server during each second of the load test scenario run, grouped by status code. HTTP status codes indicate the status of HTTP requests, for example, "the request was successful", "the page was not found";

Pages downloaded per second: Shows the number of Web pages downloaded from the server during each second of the load test scenario run. Like the Throughput graph, the Pages Downloaded per Second represents the amount of data that the VU received from the server at any given second. However, the Throughput takes into account each resource and its size (for example, the size of each .gif file, the size of each Web page);

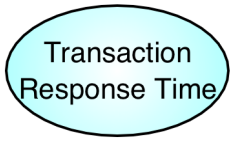
Connections: Shows the number of open TCP/IP connections at each point in time of the load test scenario;

SSL per second: Shows the number of new and reused SSL Connections opened in each second of the load test scenario. An SSL connection is opened by the browser after a TCP/IP connection has been opened to a secure server.

Object: `Transaction`
Description: Represents the Transaction metric that will be measured.
Properties:

Name (String): Metric name;

Object: `Transaction Counter`
Description: During load test scenario execution, VU generate data as they perform transactions. This metric enables collecting data that shows the transaction performance and status throughout script execution.



Properties:

Counter (String:Fixed List):

Transaction Response Time: Different response time values under different load. Average response time, maximum, percentile, and so on;

Transaction Per Second (TPS): Shows the number of transactions generated per second;

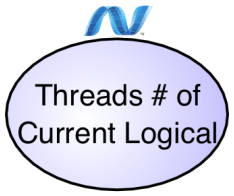
Transaction Success Rate: Shows the number of transactions that passed, failed, or stopped.

Object: `Process`
Description: Represents the Process metric that will be measured. This same metric can be associated with different process counters, *e.g.* .NET, Java, Apache, etc.

Properties:

Name (String): Metric name;

Object: `Process .NET Counter`
Description: When monitoring .NET applications, it can add .Net performance counters that cover every aspect of the Common Language Runtime (CLR) operations ranging from exception processing to security checking.



Properties:

Counter (String:Fixed List):

Exception # of Excep Thrown/Sec: Indicates the number of managed code exceptions thrown per second;

Exception Throw to Catch Depth/Sec: Indicates the number of stack frames;

Memory Large Object Heap Size: Indicates the current size of the Large Object Heap in bytes;

Memory # Bytes in all Heaps: Indicates the current memory allocated in bytes on the GC heaps;

Memory # of Pinned Objects: Indicates the number of pinned objects encountered in the last GC;

Memory % Time in GC: Indicates the percentage of elapsed time that was spent in performing a garbage collection (GC) since the last GC cycle;

Threads # of Current Logical: Indicates the number of current .NET thread objects in the application;

Threads # of Current Physical: Indicates the number of native OS threads created and owned by the CLR;

Threads # of Current Recognized: Indicates the number of threads currently recognized by the CLR;

Threads # of Total Recognized: Indicates the total number of threads recognized by the CLR since the start;

Threads Contention Rate/Sec: Indicates the rate at which threads in the runtime attempt to acquire a managed lock unsuccessfully;

Loading Current Assemblies: Indicates the number of assemblies that are loaded in the process;

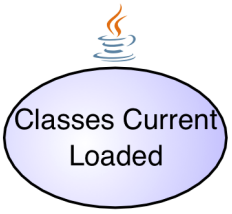
Loading Rate of Assemblies: Indicates the rate at which assemblies are loaded into the memory per second;

Loading Bytes in Loader Heap: Indicates the number of bytes committed by the class loader;

Security Total Runtime Checks: Indicates the percentage of elapsed time spent in performing runtime Code Access Security;

Security Stack Walk Depth: Indicates the depth of the stack during that last runtime Code Access Security check.

Object: `Process Java Counter`



Description: The Java platform provides comprehensive monitoring and management support. It not only defines the management interfaces for the Java Virtual Machine (JVM), but also provides out-of-the-box remote monitoring and management on the Java platform and of applications that run on it.

Properties:

Counter (String:Fixed List):

Common Uptime: Indicates how long the JVM has been running;

Common Total Compile Time: Indicates the amount of time spent in just-in- time (JIT) compilation;

Common Process CPU Time: Indicates the total amount of CPU time consumed by the JVM;

Memory Current Heap Size: Indicates the number of kilobytes currently occupied by the heap;

Memory Maximum Heap Size: Indicates the maximum number of kilobytes occupied by the heap;

Memory Committed Memory: Indicates the total amount of memory allocated for use by the heap;

Memory GC Time: Indicates the cumulative time spent on garbage collection and the total number of invocations;

Live Threads: Indicates the current number of live daemon threads plus non-daemon threads;

Peak Threads: Indicates the highest number of live threads since JVM started;

Daemon Threads: Indicates the current number of live daemon threads;

Total Started Threads: Indicates the total number of threads started since JVM started (including daemon, non- daemon, and terminated);

Current Classes Loaded: Indicates the number of classes currently loaded into memory;

Total Classes Loaded: Total number of classes loaded into memory since the JVM started, included those subsequently unloaded;

Total Classes Unloaded: Number of classes unloaded from memory since the JVM started.

Object: `Process Apache Counter`

Description: The Apache HTTP server is an open source, configurable and extensible, multi-platform Web server. In time, the Apache HTTP server became one of the most commonly used Web servers for commercial Web sites and Web-based applications.

Properties:

Counter (String:Fixed List):

CPU Load Counter: The current percentage of CPU consumed by the Apache server;

Requests Per Sec Counter: The number of requests per second (a.k.a. hits per second);

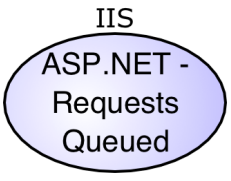
Bytes Per Sec Counter: The number of bytes transferred per second;

Bytes Per Request Counter: The number of bytes transferred per request;

Busy Workers Counter: Number of active threads serving requests;

Idle Workers Counter: Number of inactive/idle threads.

Object: `Process IIS Counter`



Description: Microsoft Internet Information Services (IIS) is the world's second most popular Web server after the Apache HTTP Server. IIS is available within all Windows operating system editions in different flavors. IIS includes the following servers: FTP/FTPS, SMTP, NNTP, and HTTP/HTTPS.

Properties:

Counter (String:Fixed List):

- *WWW Service*:

  Bytes Sent/Sec: The rate, in seconds, at which data bytes have been sent by the WWW service;

  Bytes Received/Sec: The rate, in seconds, at which data bytes have been received by the WWW service;

  Current Connections: The number of active connections to the WWW service;

Not Found Errors/Sec: The rate, in seconds, at which requests were not satisfied by the server because the requested document was not found;

Locked Errors/Sec: The rate, in seconds, at which requests were not satisfied because the requested document was locked;

Current ISAPI Extension Requests: The number of ISAPI extension requests that are being processed simultaneously by the WWW service;

ISAPI Extension Requests/Sec: The rate, in seconds, at which ISAPI extension requests are being processed by the WWW service;

- *WWW Service Cache*:

  Current File Cache Memory Usage: The number of bytes currently used for the user-mode file cache;

  Current Files Cached: The number of files whose content is currently in the user-mode cache;

  Current URIs Cached: The number of URI information blocks that are currently stored in the user-mode cache;

  Current Metadata Cached: The current number of metadata information blocks in the user-mode cache;

  Kernel: URI Cache Hits/Sec: The average number of kernel URI cache hits that are being made per second;

- *ASP.NET*:

  Requests Disconnected: The number of requests that were disconnected because a communication failure occurred;

  Requests Queued: The number of requests in the queue waiting to be serviced. If this number increases as the number of client requests increases, the Web server has reached the limit of concurrent requests that it can process. The default maximum for this counter is 5,000 requests;

  Requests Rejected: The total number of requests that were not executed because insufficient server resources existed to process them. This counter represents the number of requests that return a 503 HTTP status code, which indicates that the server is too busy;

  Errors Total/Sec: The average number of errors that occurred per second during the execution of HTTP requests. Includes any parser, compilation, or run-time errors;

  Output Cache Turnover Rate: The average number of additions to and removals from the output cache per second. If the turnover is great, the cache is not being used effectively;

  Sessions Active: The number of sessions that are active. This counter is supported only with in-memory session state.

  Transactions/Sec: The average number of transactions that were started per second;

  Transactions Pending: The number of transactions that are in progress;

- *Active Server Pages (ASP)*:

Errors/Sec: The average number of errors that occurred per second;

Requests/Sec: The average number of requests that were executed per second;

Requests Executing: The number of ASP requests currently executing (for example, the number of active worker threads);

Requests Queued: The number of queued ASP requests that are waiting to be processed. The maximum number for this counter is determined by the metabase property AspRequestQueueMax;

Transactions/Sec: The average number of transactions that have been started, per second.

Object: `Process WebSphere Counter`

Description: The IBM WebSphere Application Server is the flagship product in the IBM WebSphere platform. It is one of the top J2EE application servers. WebSphere architecture and infrastructure are oriented for performance and scalability, and allow deployment of many types of distributed applications such as Web-based applications and Web services. WebSphere Application Server provides a Performance Monitoring Infrastructure (PMI) which is a server side monitoring infrastructure that offers client-side API. Using PMI can monitor the overall health and performance of the application server. The performance data is made available via JMX.

Properties:

Counter (String:Fixed List):

- *Enterprise Java Beans (EJB)*:

  Ready Counter: The number of concurrently ready beans (entity and session);

  Live Counter: The number of concurrently live beans;

  Method Response Time: The average response time (in milliseconds) on the bean methods (home, remote, local);

  Active Method Counter: The number of concurrently active methods - the number of methods called at the same time;

  Message Counter: The number of messages delivered to the bean onMessage method (message driven beans);

  Message Backout Counter: The number of messages that failed to be delivered to the bean onMessage method (message driven beans);

  Pooled Counter: The number of objects in the pool (entity and stateless);

  Wait Time: The average time taken to obtain a ServerSession from the pool (message driven bean).

- *JDBC Connection Pool*:

  Concurrent Waiters: The number of threads that are currently waiting for a connection;

  Faults: The total number of faults, such as timeouts, in the connection pool;

  Percent Used: The average percent of the pool that is in use.

- *Java Virtual Machine (JVM)*:

Free Memory: The free memory in the JVM run time;
Process CPU Usage: The CPU Usage (in percent) of the Java virtual machine;
Used Memory: The used memory in the JVM run time.

- *Servlet Session*:
  Active Count: The number of concurrently active sessions. A session is active if the WebSphere Application Server is currently processing a request;
  Live Count: The number of local sessions that are currently cached in memory.
- *Transaction*:
  Active Counter: The number of concurrently active global transactions;
  Local Active Counter: The number of concurrently active local transactions;
  Rolledback Counter: The total number of global transactions rolled back;
  Local Rolledback Counter: The number of local transactions rolled back;
  Global Timeout Counter: The number of global transactions timed out;
  Local Timeout Counter: The number of local transactions timed out.
- *Thread Pool*:
  Active Counter: The number of concurrently active threads;
  Pool Size: The average number of threads in pool;
  Percent Maxed: The average percent of the time that all threads are in use;
  Declared Thread Hung Counter: The number of threads declared hung.
- *Web Application*:
  Concurrent Requests: The number of requests that are concurrently processed;
  Service Time: The response time (in milliseconds) of a servlet request;
  Concurrent Requests: The number of requests processing concurrently for a URI associated with a servlet;
  Service Time: The average service response time (in milliseconds) for a URI associated with a servlet.
- *System*:
  CPU Usage Since Last Measurement: The average system CPU utilization taken over the time interval since the last reading;
  Free Memory: The amount of real free memory available on the system.

Object: `Process Oracle Database Counter`

Description: The Oracle database is a relational database management system (RDBMS) produced by the Oracle Corporation. The Oracle database is rich with features that contribute to its high availability, scalability, performance, manageability, and security. These features make Oracle an enterprise class RDBMS and one of the top leaders in this realm. The Oracle database has comprehensive support for application development owing to different capabilities and features. Oracle also offers data access methods for both Java and .NET.

Properties:

Counter (String:Fixed List):

Sorts (disk): Number of sort operations that required at least one disk write. Sorts that require I/O to disk are quite resource intensive;

Sorts (memory): Number of sort operations that were performed completely in memory and did not require any disk writes. Sorting is usually caused by selection criteria specifications within table join SQL operations.

Db block gets: Number of blocks accessed in buffer cache for INSERT, UPDATE, DELETE, and SELECT FOR UPDATE. Represent block logical reads (from cache). The logical reads ALWAYS include the physical reads. Low number of physical reads is preferable;

Consistent gets: Number of blocks accessed in buffer cache for normal queries (SELECTs without for update clause). Represent block logical reads (from cache). The logical reads ALWAYS include the physical reads. Low number of physical reads is preferable;

Physical reads: Total number of data blocks read from disk. This number equals the value of physical reads direct plus all reads into buffer cache. Low number of physical reads is preferable. This number must be compared to logical reads to calculate cache hit ratio. Logical reads is the sum of database block gets and consistent gets;

Physical writes: Total number of data blocks written to disk. This number equals the value of physical writes direct plus all writes from buffer cache;

Redo writes: Total number of writes by LGWR to the redo log files. redo blocks written divided by this statistic equals the number of blocks per write;

Redo entries: Redo entries contain the information necessary to reconstruct, or redo, changes made to the database by INSERT, UPDATE, DELETE, CREATE, ALTER, or DROP operations. Redo entries are used for database recovery, if necessary. Redo entries -> successful redo writes. Ratio Redo buffer allocation retries / Redo entries should be less than 1%;

Redo buffer allocation retries: Total number of retries necessary to allocate space in the redo buffer. Retries are needed either because the redo writer has fallen behind or because an event such as a log switch is occurring. Redo buffer allocation retries -> failed redo writes. Ratio Redo buffer allocation retries / Redo entries should be less than 1%;

Redo log space requests: Number of times the active log file is full and Oracle must wait for disk space to be allocated for the redo log entries. Such space is created by performing a log switch. Log files that are small in relation to the size of the SGA or the commit rate of the work load can cause problems. When the log switch occurs, Oracle must ensure that all committed dirty buffers are written to disk before switching to a new log file;

Parse count (hard): Total number of parse calls (real parses). A hard parse is a very expensive operation in terms of memory use, because it requires Oracle to allocate a workheap and other memory structures and then build a parse tree. Should be minimized. The ratio of Hard Parse to Total should be less than 20%;

Parse count (total): Total number of parse calls (hard and soft). A soft parse is a check on an object already in the shared pool, to verify that the permissions on the underlying object have not changed. The ratio of Hard Parse to Total should be less than 20%;

Parse time CPU: Total CPU time used for parsing (hard and soft) in 10s of milliseconds;

Parse time elapsed: Total elapsed time for parsing, in tens of milliseconds. Subtract parse time cpu from this statistic to determine the total waiting time for parse resources;

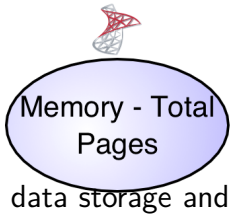
CPU used by this session: Amount of CPU time (in tens of milliseconds) used by a session from the time a user call starts until it ends. If a user call completes within 10 milliseconds, the start- and end-user call times are the same for purposes of this statistic, and 0 milliseconds are added;

*Bytes sent via SQL*Net to client*: Total number of bytes sent to the client from the foreground processes. Gives a general indication regarding the amount of data transferred over the net;

*Bytes received via SQL*Net from client*: Total number of bytes received from the client over Oracle Net Services. Gives a general indication regarding the amount of data transferred over the net;

Logons current: Total number of current logons.

Object: `Process MS SQL Server Database Counter`



Description: Microsoft SQL Server is one of the most widely used database systems. It has grown from handling small departmental tasks to serving up the largest databases on the planet. No longer a simple "database", Microsoft SQL Server is now a complete data architecture solution capable of handling the data storage and manipulation needs of any organization. Organizations can use this solution to store and manage many types of data, including XML, email, time/calendar, file, document, geospatial, and so on, while providing a rich set of services to interact with the data: search, query, data analysis, reporting, data integration, and robust synchronization. Developers can write applications that access SQL Server from a server to a desktop or mobile device using a variety of technologies, whether Microsoft based or third party vendors. SQL Server is available in many editions to help meet the needs of any organization. From Express and Compact to Workgroup to Standard and Enterprise, each edition delivers sets of features targeted to specific needs while maintaining the same level of functionality for developers and end users. It used to be said that SQL Server works great right out of the box and performance is never an issue. However, the advent of cheaper hardware and the explosion of data is pushing more users against the limits of the out-of-the-box performance of SQL Server. It is the job of the performance engineer to find these problems by using various monitoring techniques.

Properties:

Counter (String:Fixed List):

- *CPU*:

  SQL Compilations/Sec: Indicates the number of times compilations occurred per second;

SQL Re-Compilations/Sec: Indicates the number of times re-compilations occurred per second;

Batch Requests/Sec: Indicates the number of Transact-SQL command batches received per second.

- *Memory*:

  Total Pages: Indicates the number of pages in the buffer pool. Target Pages: Indicates the ideal number of pages in the buffer pool;

  Total Server Memory (KB): Indicates the amount of memory the KB SQL Server is currently using;

  Target Server Memory (KB): Indicates the amount of memory the KB SQL Server needs to operate efficiently.

  Buffer Cache Hit Ratio: Indicates the percentage of pages that were found in the memory;

  Page Life Expectancy: Indicates the number of seconds a page will stay in the buffer pool without reference;

  Stolen Pages: Indicates the number of pages used for miscellaneous server purposes (including procedure cache);

  Cache Hit Ratio: Indicates the ratio between cache hits and lookups;

  Memory Grants Pending: Indicates the total number of processes waiting for a workspace memory grant;

  Checkpoint Pages/Sec: Indicates the number of pages flushed to disk per second by a checkpoint or other operation that requires all dirty pages to be flushed;

  Lazy Writes/Sec: Indicates the number of buffers written per second by the buffer manager's lazy writer.

- *Disk*:

  Full Scans/Sec: Indicates the number of unrestricted full scans per second;

  Page Splits/Sec: Indicates the number of page splits per second that occur as a result of overflowing index pages;

  Temp Tables Creation Rate: Indicates the number of temporary tables/table variables created per second;

  Locks - Average Wait Time (ms): Indicates the average amount of wait time (in milliseconds) for each lock request that resulted in a wait.

Relationship: `Association Counter Criteria Threshold`

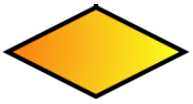
Description: Binds a Process object on a pair of targets: a Criteria object and a Threshold object. It is important to mention that the cardinality of the connection Criteria object is minimum 1 and maximum 2. The maximum cardinality is applied when the Between association criteria property is set.

Properties:

Association Criteria (String:Fixed List):

Between: When a performance counter binds with two Criteria objects;
Greater than or equal to: Used when the value of Threshold object is greater than or equal to the performance counter for the established criterion (Number of Virtual Users);
Less than: Used when the value of Threshold object is less than the performance counter for the established criterion (Number of Virtual Users).

Relationship: `Association Metric Counter`
Description: Binds a Metric object on a Counter object.
Properties: There are none.



A.2 Canopus Performance Scenario Specification

Object: `User Profile`
Description: A user profile represents a set of virtual users, in which similar interations with the SUT are performed.
Properties:
Name (String): User profile name;
Description (Text): A textual note about the user profile description;
Percentage (Number): If there are more than one User Profile objects in a `Canopus Percentage Scenario` model, a percentage must be attributed to every one of these profiles, *i.e.* it defines the distribution of number of users that will execute a test in accordance with their user profile. By default, if there is just one User Profile object, the percentage is set to 100%.

Object: `Script`
Description: A test Script object represents a set of activities performed by a User Profile object. This Script object allows a `Canopus Performance Scripting` model to be associated, *i.e.*, this object can decompose in a sub-graph. This object refers to the `Canopus Performance Scripting` model, *i.e.* each Script object is detailed in a subgraph, being enriched with other performance information, such as activities, transactions, thinktime, test data, etc.
Properties:
Name (String): Test script name;
Description (Text): A textual note about the test script description.

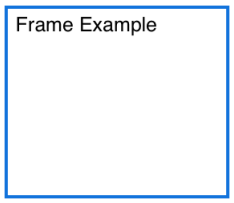
Object: `Workload`
Description: A `Canopus Performance Scenario` allows setting one or more Workload objects. A Workload object represents a workload profile which is

composed of several workload objects, such as number of VU, ramp up users and time, ramp down users and time, and test duration.

Properties:

Workload (CanopusPerformanceWorkload metamodel): Allows to associate with a `Canopus Performance Workload` model (see Section A.2.1). The name of the associated workload is rendered on the right side of the "hourglass" icon, *e.g.* Load Testing.

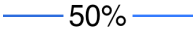
Object: `Frame`



Description: Used to visually group other objects that compose the `Canopus Performance Scenario` model. The model name is rendered on the top-left side of the frame;

Properties: There are none.

Relationship: `Association`



Description: Connects a User Profile object to a Script object.

Properties:

Percentage (Number): If a User Profile object is associated with more than one test Script object, a percentage must be attributed to every association from the script belonging to this user profile, *i.e.* it defines the distribution of the number of users that will execute a test script. By default, if a user profile is associated with just one Script object, the percentage is set to 100%. In this case, the percentage is hidden with just the association line being rendered.

A.2.1 Canopus Performance Workload Specification

Object: `Virtual Users`

Description: Represents the total number of virtual users who will make requests to the SUT for a given performance workload scenario.

Properties:

Virtual Users (Number): Determines the number of virtual users who will make requests to the SUT.

Object: `Test Duration`

Description: Refers to the total time of performance test execution for a given workload. This duration time includes total ramp up time and total ramp down time.

Properties:

Time (String): Determines the total time of performance test execution for a given `Canopus Performance Workload` model. It is defined in the format (hh:mm:ss:ms).

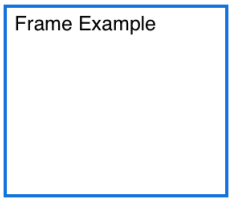
Object: `Ramp Up User`
Description: Represents the number of virtual users who will access the SUT during each ramp up time interval.
Properties:
Virtual Users (Number): Determines the number of virtual users who will access the SUT.

Object: `Ramp Up Time`
Description: Represents the time it takes for each set of ramp up users to access the SUT.
Properties:
Time (String): Determines the time defined in the format (hh:mm:ss:ms).

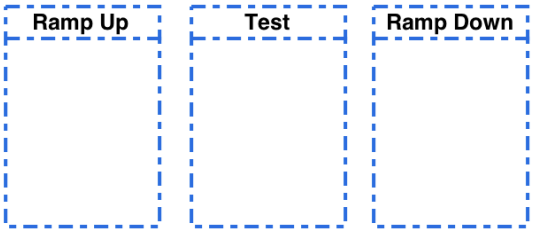
Object: `Ramp Down User`
Description: Represents the number of virtual users who will leave the SUT on each ramp down time.
Properties:
Virtual Users (Number): Defines the number of virtual users who will leave the SUT on each ramp down time.

Object: `Ramp Down Time`
Description: Defines the time it takes for a given ramp down user stop the testing.
Properties:
Time (String): Determines the time defined in the format (hh:mm:ss:ms).

Object: `Frame`
Description: Used to visually group other objects that compose the `Canopus Performance Workload` model. The model name is rendered on the top-left side of the frame;
Properties: There are none.



Object: `Workload Group`
Description: A Workload Group is a simple box rendered for grouping workload objects in a `Canopus Performance Workload` model.
Properties: There are none.



A.3 Canopus Performance Scripting Specification

Object: `Initial`

Description: Determines the start of the interaction of a virtual user within a test script. Every script must have only one Initial object. The graphical notation for the Initial object is shown as a double circles.

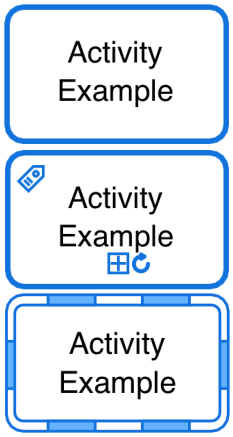
Properties: There are none.

Object: `Final`

Description: Defines the end of the interaction of a virtual user within a test script. Every script must have only one Final object. The graphical notation for the Initial object is shown as a thick single circle.

Properties: There are none.

Object: `Activity`



Description: Characterizes the interactions of virtual user behavior from the point of view of the SUT. Every script must have at least one Activity object. This object allows to repeat not only one but a set of activities, it is possible to associate a subgraph model (another `Canopus Performance Scripting`) into an Activity object. This feature is represented by "+" (plus symbol) on the bottom of the Activity object.

Properties:

Name (String): Activity name;

Action (String): HTTP address of the SUT request;

Method (String:Radio Button Set): HTTP method, *i.e.* GET or POST;

Type Action (Fixed List): Defines the type of HTML elements that a user will interact, such as: Body, Button, Checkbox, Drop-down list, Input field, Image, Link, Radio button, Save Parameter, Submit button, Textarea;

Parameters (Collection:Parameter): One or more Parameter object can be associated to an Activity object. It is possible to create a new or add an existing Parameter object already previously pre-defined. When at least one Parameter object is associated, a "Tag" icon is rendered on the top-left side of the Activity object;

Loop Instances (Number): Defines the number of repetitions of the activity. By default, the standard value is 1. When a number is more than one, a "Loop" icon is rendered on the bottom of the Activity object;

Transaction Activity (Transaction): Allows to define activities that will be controlled by their transactions. There are two transaction response performance counters: Response Time Seconds and Response Size Bytes. Both are defined by number values. When a Transaction object is associated with an Activity object, the border of activity becomes dashed.

Object: `Think Time`

Description: Denotes the time between the moment the activity becomes available to the user and the moment the user decides to execute it, for example, the time of filling in form before its submission.

Properties:

Name (String): Think time name;

Time (String): Defined in the format (hh:mm:ss:ms). This time value is rendered on the bottom of the Think Time object.

Object: `Save Parameters`

Description: Supports that a parameter generated in runtime can be saved to be used in other activities of the same script flow.

Properties:

Name (String): Save parameters name. This name is rendered on the right side of the Save Parameters object;

Parameters (Collection:Parameter): One or more Parameter objects can be associated to an Save Parameters object. It is possible to create a new or add an existing Parameter object already previously pre-defined.

Object: `Data Table`

Description: Determines the data that is consumed by the activities. Specified by filename that must contain a data table, *e.g.* a .CSV file.

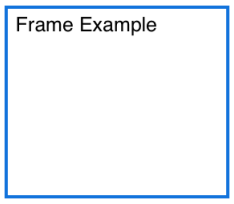
Properties:

Attached File (String: External Element): File name;

Delimiter (String): Specifies the delimiter among file data, *e.g.* ";" (semicolon);

Parameter List (Collection:Columns): Sets the data tables parameter list (fields or columns).

Object: `Frame`



Description: Used to visually group other objects that compose the `Canopus Performance Scripting` model. The test script name is rendered on the top-left side of frame;

Properties: There are none.

Object: `Parameter` (Hidden Object)

Description: Determines the data that is consumed by the activities. Specified by a filename that must contain a data table, *e.g.* a .CSV file.

Properties:

Type Action (Fixed List): Defines the type of HTML elements that a user will interact, such as: Body, Button, Checkbox, Drop-down list, Input field, Image, Link, Radio button, Save Parameter, Submit button, Textarea;

Parameter Name (String): Parameter name of the SUT. For instance, the property name of a HTML form element;

Static Value (String): Static value defined for the parameter;

Dynamic Value (Columns): Allows to define a dynamic value base on a Columns object.

Object: `Columns` (Hidden Object)

Description: Defines the strategy that each column from the test data will be randomized.

Properties:

Parameter Name (String): Name of dynamic parameter;

Updated Value On (String: Fixed List): Specifies the update method for a Columns object that will associate with a Parameter object.

EachInteraction: It instructs the VU to use a new value for each iteration;

EachOccurence: It instructs the VU to use a new value for each occurrence of the parameter;

Once: It instructs the VU to update the value of the parameter only once during the execution.

Select Next Row (String: Fixed List): Defines the way the data should be selected for the parameter during execution:

Sequential: The parameter will hold the same value during the same iteration.

Random: A value randomly selected;

Unique: The parameter will hold a unique value for each virtual user. When this option is selected the total rows of data should at least be as many as the total virtual users;

SameAsOtherParam: The data row for the parameter can be aligned with that of another parameter. For instance, suppose we have two parameters, one for the user name (pUsr) and one for the password (pPwd) then the next row selected for the password will be set to the same line as "pUsr" parameter.

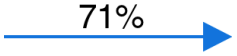
Referended Parameter (String): Defines the referenced parameter name when the "SameAsOtherParam" option is selected in the Select Next Row property.

Relationship: `Association`

Description: Connects an Activity object to a Data Table. It also allows to connect an Activity object to a Save Parameters object in dual directions.

Properties: There are none.

Relationship: `Transition`

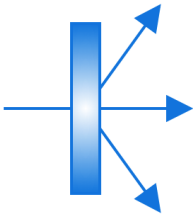


Description: The main binding, which connects to a different pair of objects, such as: from an Initial to an Activity; from an Activity to another Activity or a Final or a Think Time object; and, from a Think Time to an Activity. It is determining the flow iterations performed by the virtual user to the SUT.

Properties:

Percentage (Number): If an object is associated with more than one Transition relationship, a percentage must be attributed to every Transition relationship from this object to sum 100%, *i.e.* it defines the distribution of the number of virtual users that will execute each of the derived test scripts. By default, if a Transition relationship is associated with just one object, the percentage is set to 100%. In this case, the percentage is hidden, with only the transition line being rendered.

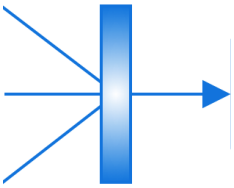
Relationship: `Fork`



Description: It is a control object that has one incoming edge and multiple outgoing edges and is used to split incoming flow into multiple concurrent flows. Fork objects are introduced to support parallelism in activities. The graphical notation for a Fork object is a line segment with a single Transition edge entering it, and two or more edges leaving it.

Properties: There are none.

Relationship: `Join`



Description: It is a control object that has multiple incoming edges and one outgoing edge and is used to synchronize incoming concurrent flows. Join objects are introduced to support parallelism in activities. The graphical notation for a Join object is a line segment with several Transition edges entering it, and only one edge leaving it.

Properties: There are none.

A.3.1 Canopus Performance External File Specification

Object: `External File` (Hidden Object)

Description: External file that is loaded to be associated with a Data Table object.

Properties:

Name (String): Specifies a unique location in a file system, including its extension.

APPENDIX B – CASE STUDY MODELS

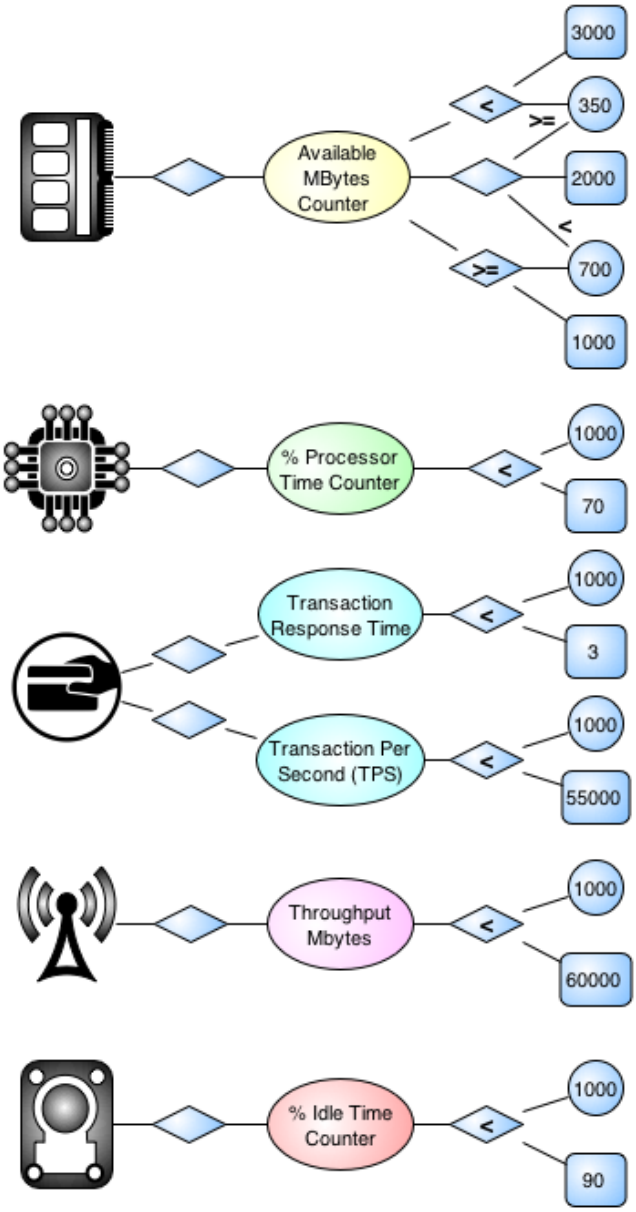


Figure B.1: Graphical representation of the Changepoint `Canopus Performance Metric` model

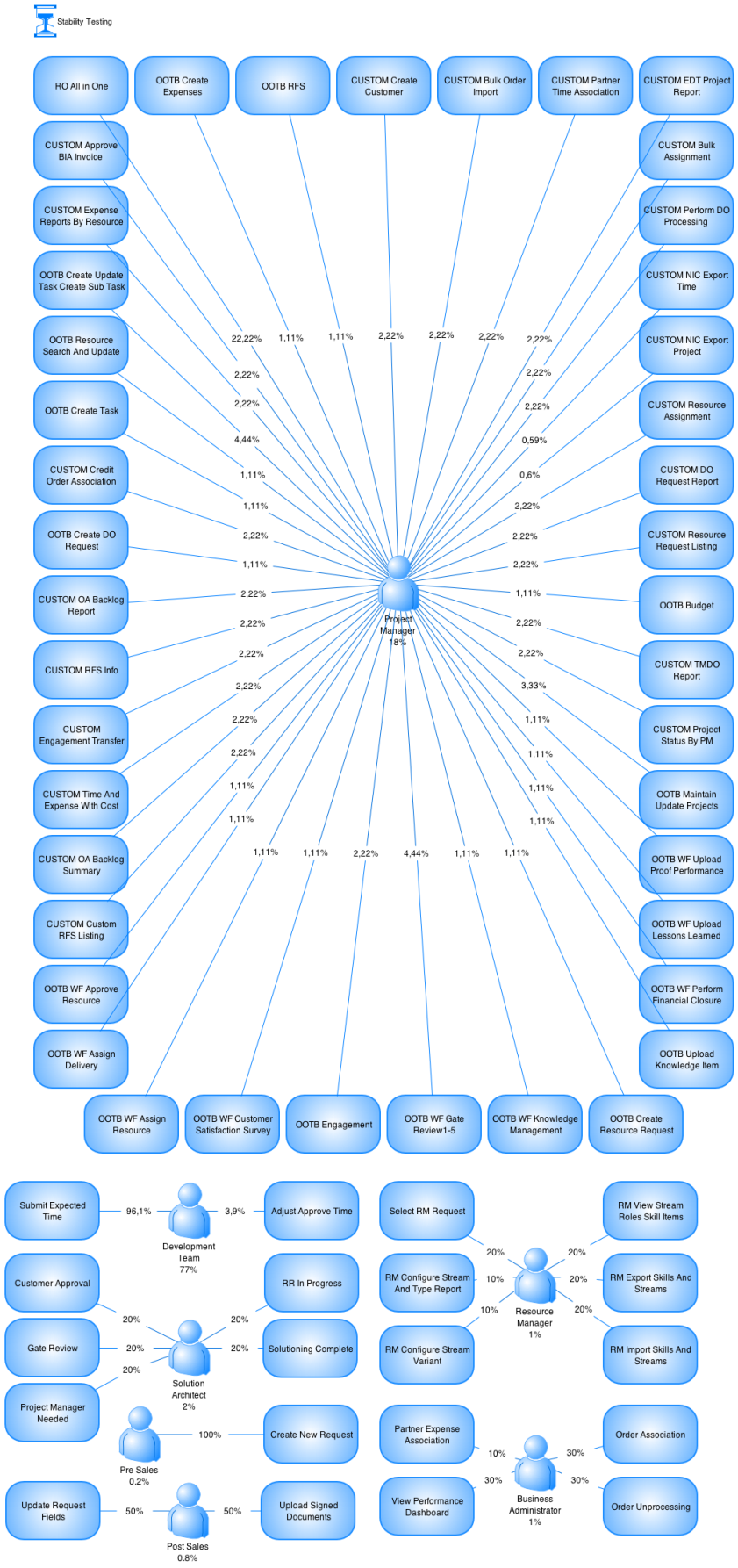


Figure B.2: Graphical representation of the Changepoint `Canopus Performance Scenario` model

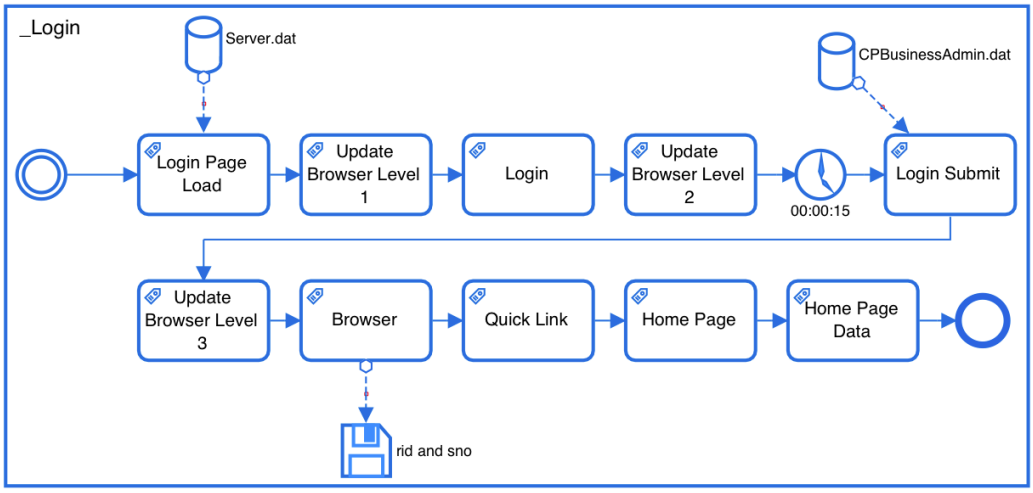


Figure B.3: Graphical representation of the Changepoint `Login` Canopus `Performance Scripting` model

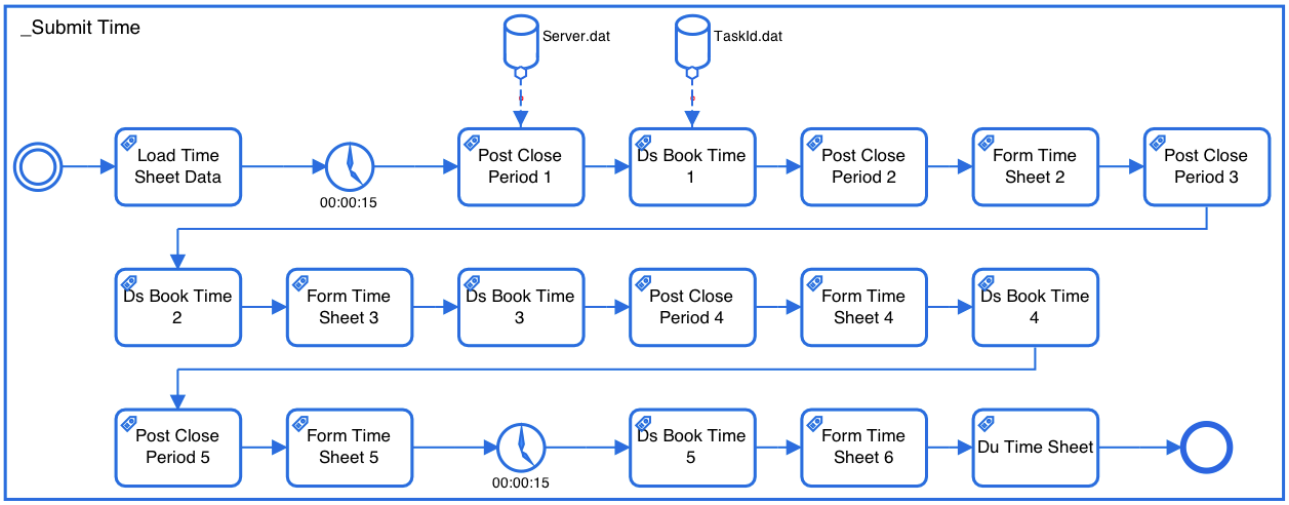


Figure B.4: Graphical representation of the Changepoint `Submit Time` Canopus `Performance Scripting` model

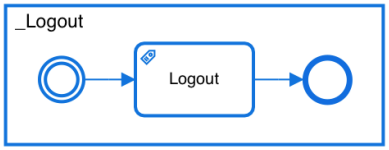


Figure B.5: Graphical representation of the Changepoint `Logout` Canopus `Performance Scripting` model

APPENDIX C – A SURVEY TO EVALUATE THE DOMAIN-SPECIFIC LANGUAGE FOR MODELING PERFORMANCE TESTING

A Survey to evaluate the Domain-Specific Language for Modeling Performance Testing

Questionnaire Instrument

Introduction

We would like to invite you to participate in a research study of DomainSpecific Language (DSL) for performance testing modeling. The study intends to evaluate the quality of graphical elements of our DSL, as well as their representativeness regarding the performance testing modeling. The main idea is generate performance scenarios and scripts based on the artifacts modeled with our DSL.

It is recommended that you have some expertise or knowledge in performance testing to complete this survey. Please read this form and ask any question you may have before agreeing to participate in this study.

This study is being conducted by: Maicon Bernardino, Doctoral student at Pontifical Catholic University of Rio Grande do Sul (PUCRS), advised by Avelino Francisco Zorzo.

Background Information: The purpose of this survey is to evaluate the graphical elements and their representativeness to symbolize performance elements that compose a DSL for performance testing modeling.

Procedures: If you agree to participate in this study, we ask you to complete 1 survey based on an extensive literature review. You will be asked also to identify the activities of performance testing profiles, and identify the needed performance elements to symbolize the main performance artifacts (e.g., scenarios, scripts.). Most of the questions are multiple choice, but some of them have a field for suggestions or comments. It will take, you between 15 to 30 minutes to complete the survey.

Risks: Being a participant in this study has no foreseeable risks.

Benefits: The researcher hopes to develop a DSL for performance testing modeling that use the evaluated graphical elements. We way also include suggested elements in out DSL. The researcher believes that our DSL may benefit several testers around the world to be more effective when describing performance scenarios and scripts. As a participant, you will have access to the results of this research.

Confidentiality: The records of this study and the secured files will be kept private. In any sort of report we might publish them not including any information which will make it possible to identify a subject. Only the primary researcher will have access to the records.

Contact and Questions: The researcher conducting this study is Maicon Bernardino. Please contact him with any question:

bernardino@acm.org

If you wish to participate, please start the survey by clicking in the following URL.
http://www.cepes.pucrs.br/survey/

Profile Questions

SQ1 What is the highest level of education you have completed? **Please select one answer below.**

- Associate degree (for example: AA, AS)
- Bachelor's degree (for example: BA, AB, BS)
- Master's degree (for example: MA, MS, MEng, MEd, MSW, MBA)
- Professional degree (for example: MD, DDS, DVM, LLB, JD)
- Doctorate degree (for example: PhD, EdD)
- Postsecondary diploma
- Postsecondary certificate
- Other

SQ2 How many years of experience do you have in the performance testing area? **Please select one answer below.**

- 14+
- 11-13
- 8-10
- 5-7
- 2-4
- 0-1 year

SQ3 What is your main job position? **Please select one answer below.**

- Performance Test Manager
- Performance Test Engineer
- Performance Test Analyst
- Performance Tester
- Quality Assurance / Audit Manager
- Software Engineering / Developer
- Project Manager
- Program Manager
- IT Manager
- CEO
- Researcher
- Other

SQ4 Which one of the following best describes your organization? **Please select one answer below.**

- Privately held, forprofit business
- Publicly held, forprofit business
- Notforprofit service organization
- Primary or Secondary School
- Community/Technical College
- Bachelor's College
- Master's Comprehensive College
- Doctoral/Research University
- Other

SQ5 In your performance testing practice, do you use some kind of model, notation or language to generate, represent or design your performance scenarios and scripts? If so, can you describe them? **Please write them in the space below**.

SQ6 Finally, please indicate your name, your organization's name and your email address below if you would like to receive a summary of the results. Your data will be combined with the data of other respondents and shared only in aggregate.
SQ6.1 Name: _____
SQ6.2 Name of organization: _____
SQ6.3 Email address: _____
SQ6.4 Please check here if we may contact you for further information or if you would like to be the subject of a case study.

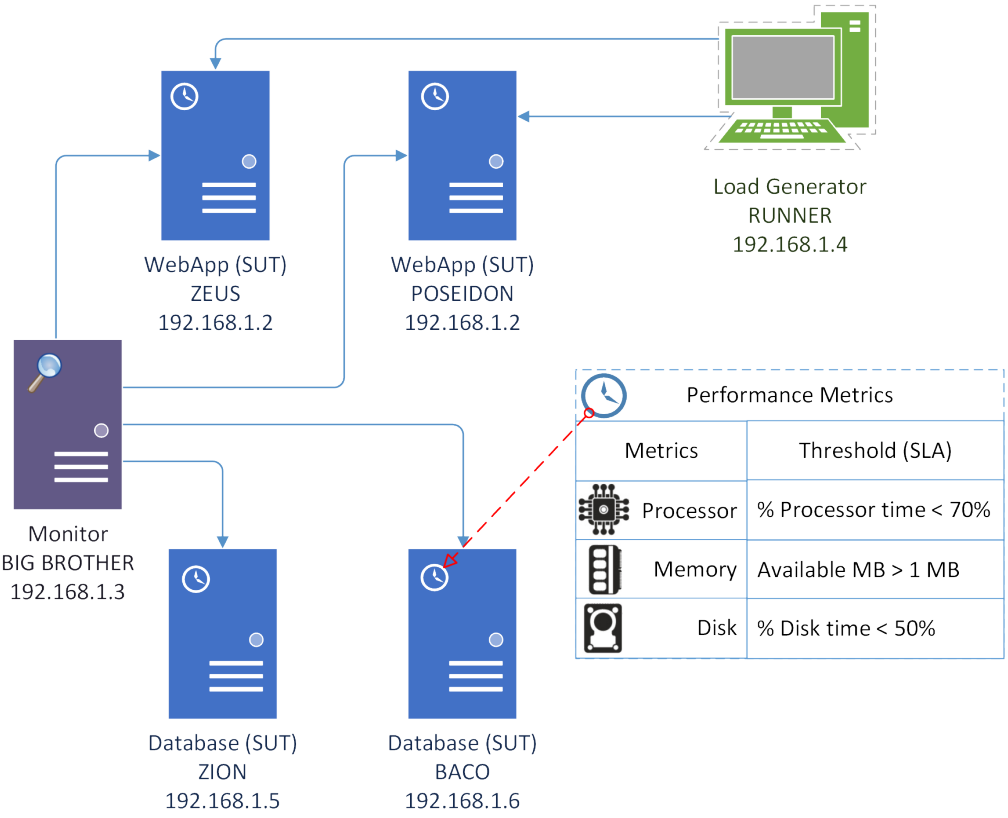
DSL Questions

Our DSL is composed of three parts: monitoring, scenario, and scripting.

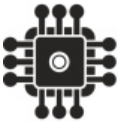
Scale: **DC** *Disagree Completely;* **DS** *Disagree Somewhat*; **NN** *Neither Agree Nor Disagree*; **AS** *Agree Somewhat*; **AC** *Agree Completely*

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 hostname. 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 in to 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 monitor the performance counters of the SUT.



SQ7 Please indicate how much you agree or disagree with each of the following monitoring elements that compose our monitoring diagram for performance testing. **Please check one option for each item listed below**.

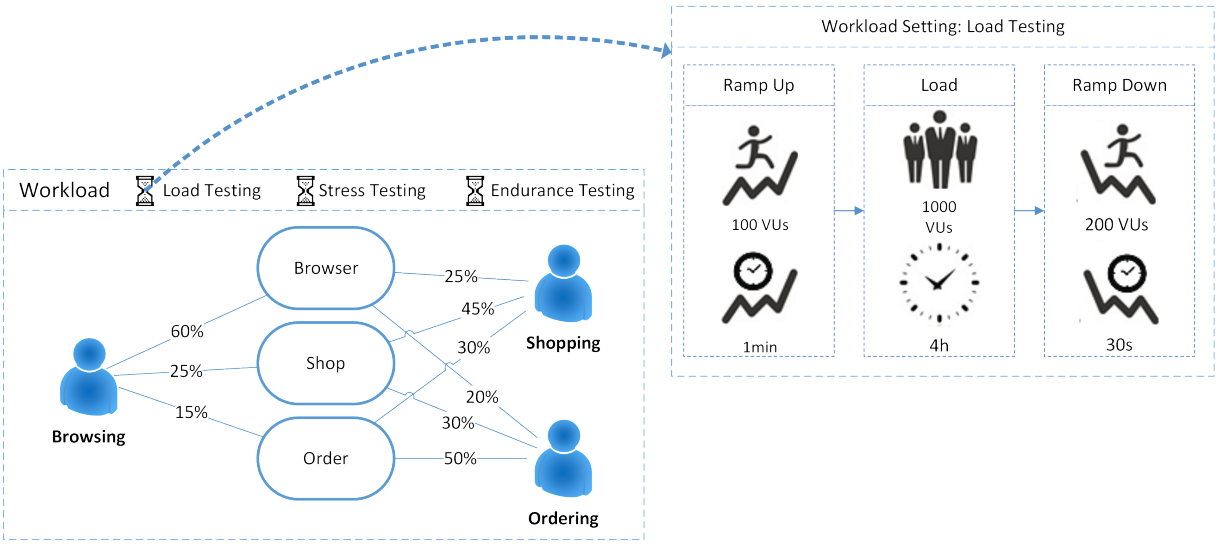
Element	Image	Description	DC	DS	NN	AS	AC
SUT		System Under Test					
Load Generator		Responsible to generate workload for SUT.					
Monitor		Responsible by monitoring of performance metrics.					
Memory		Memory performance metric.					
Processor		Processor performance metric.					
Process		Process performance metric, such as Apache, Java, etc.					
Disk		Disk performance metric.					
Network Interface		Network interface performance metric.					
Throughput		Rate at which requests are met.					

Element	Image	Description	DC	DS	NN	AS	AC
Transaction		Monitor the specified transactions into performance scripts.					
Response Time		Monitor the response time, i.e., the elapsed time between the client request and the server answer, including the network latency.					
Response Size		Number of bytes returned in a response of the SUT.					
Flow		Determine the flow among elements into a performance monitoring diagram.					

SQ8 Would you like to add some unspecified element or remove some presented element? If so, which ones and why? **Please write your answer in the space below**.

Scenario

The performance scenario part allows to set 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 probability is attributed between the user profile and each test script, i.e. it describes the probability that that test script is executed. In addition to setting user profiles, in this part it also important to set one or more workload profiles.



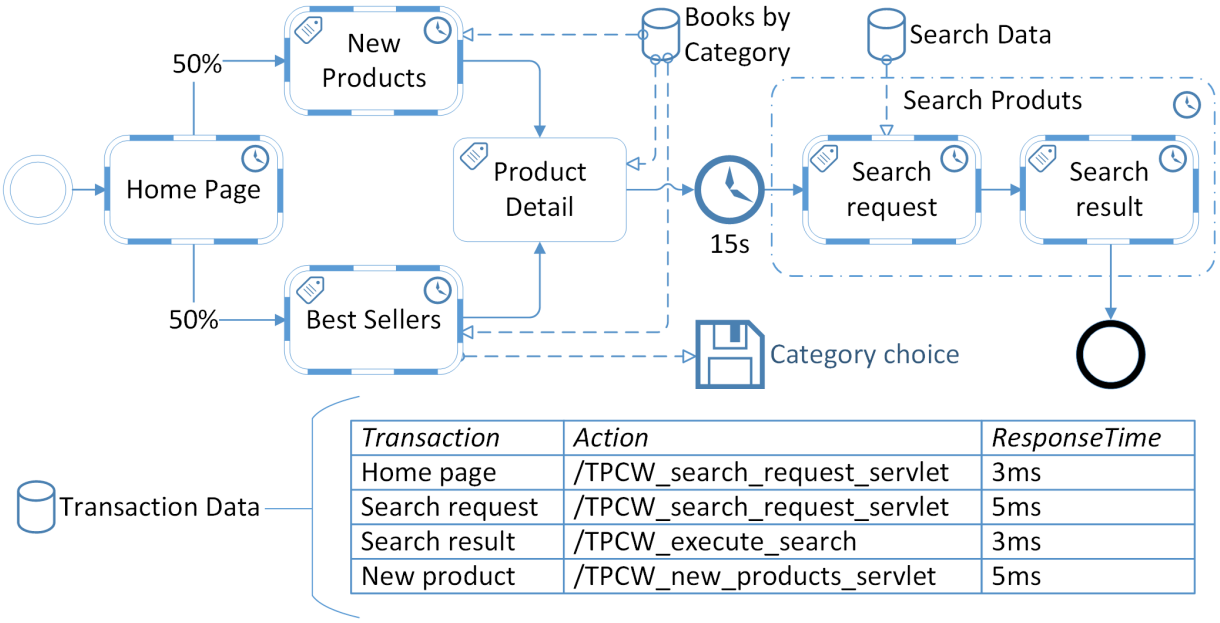
SQ9 Please indicate how much you agree or disagree with each of the following scenario elements that compose our scenario diagram for performance testing. **Please check one option for each item listed below**.

Element	Image	Description	DC	DS	NN	AS	AC
User Profile		User profile					
Script		It determines a model iteration user profile with the activities performed by each virtual user. This element refers to a performance script diagram.					
Association		Responsible to associate the user profile and script elements.					
Workload		Characterize the different workload of user profiles.					
Ramp Up Time		Time it takes for each set of ramp up users to access the SUT.					
Ramp Up User		Number of VUs who will access the SUT during each ramp up time interval.					
Ramp Down Time		Defines the time it takes for a given ramp down user stop the testing.					
Ramp Down User		Defines the number of VUs who will left the SUT on each ramp down time.					
Virtual User		Number of VUs who will make requests to the SUT.					
Test Duration		Refers to the total time of performance test execution for a given workload.					

SQ10 Would you like to add some unspecified element or remove some presented element? If so, which ones and why? **Please write your answer in the space below**.

Scripting

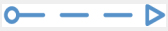
The performance script part represents each of the test scripts from the user profiles in the scenarios part. This part is responsible for determining the behavior of the interaction between VUs and SUT. Each test script includes activities such as transaction control or think time between activities. The same way as there is a probability 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. During the description of each test script it is also possible to define a decision table associated to each activity. This decision table represents the decisions that is taken by a user based on the response that an application provides. Actually, the decision table will be used to guarantee that the user distribution rate is achieved.

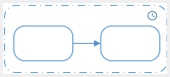


Transaction	*Action*	*ResponseTime*
Home page	/TPCW_search_request_servlet	3ms
Search request	/TPCW_search_request_servlet	5ms
Search result	/TPCW_execute_search	3ms
New product	/TPCW_new_products_servlet	5ms

SQ11 Please indicate how much you agree or disagree with each of the following script elements that compose our script diagram for performance testing. **Please check one option.**

Element	Image	Description	DC	DS	NN	AS	AC
Initial		Determines the start execution of a script. Every script must have an Initial element.					

Element	Image	Description	DC	DS	NN	AS	AC
Final		Determines the end execution of a script. Every script must have a Final element.					
Activity		Characterizes the interactions from the point of view of the behavior of the Virtual User with the SUT. Every script must have at least one Activity element.					
Transaction Activity		Specifies Activity elements that will be monitored. Visually, the activity is displayed with a dashed border. Every Transaction Activity controls an atomic activity, i.e. a single activity.					
Data Table		Determines the data that is consumed by Activity elements. Specified by filename that should contain a table with the test data, e.g. a file with .CSV extension.					
Association		Binds the elements consumed by other elements. For instance, the Activity element may consume test data represented by Data Table element.					
Transition		Associates the elements of Activities, determining the flow of iterations performed by a Virtual User element.					
ThinkTime		Denotes the idle time between each iteration of the Virtual User with the SUT, such as the time of filling out a registration form until data submission. Must be defined outside of Transaction elements and can be annotated between each pair of Activity elements.					
Save Parameters		Parameters stored after a response from the SUT so that your information can be used later in another Activity or Decision Table elements.					

Element	Image	Description	DC	DS	NN	AS	AC
Parameters		Linked to the test data Data Table element, the Parameters are specified in each of the Activity elements that consume the test data during the execution of the performance test scripts. Graphically, an activity has parameters associated with it when it presents in the top left corner of the activity a tag symbol.					
Decision Table		Symbolized in the bottom right corner of the Activity element by a diamond. It's responsible for determining the variability of flow iteration by Virtual User elements in a Performance Script diagram.					
Group		Allows you to group a series of Activity elements. For instance, you can measure a sequence of activities that match a complete job from the standpoint of business, i.e. one Transaction consisting of a set of subtransactions.					
Concurrent		Makes it possible to run multiple instances of an activity in parallel/concurrent.					
Loop		Lets you run multiple instances of an Activity sequentially.					

SQ12 Would you like to add some unspecified element or remove some existing element? If so, which ones and why? **Please write your answer in the space below**.

APPENDIX D – EXPERIMENT INTRUMENTS

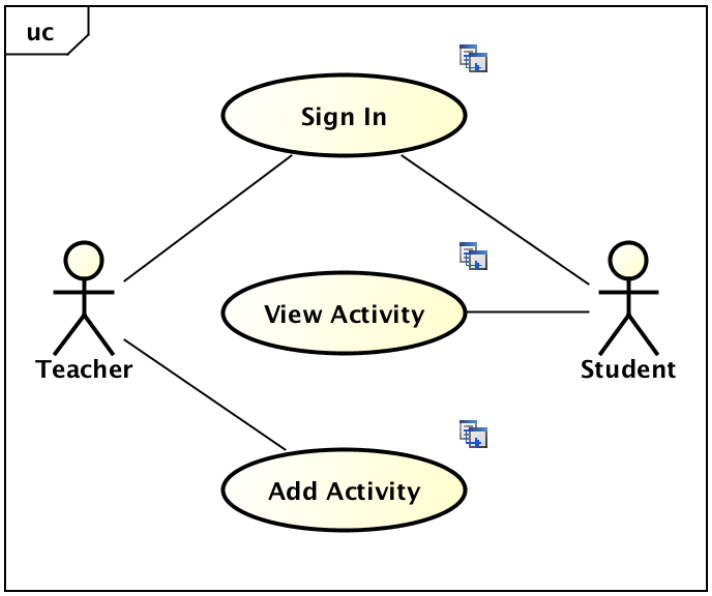


Figure D.1: UML use case diagram of the Moodle

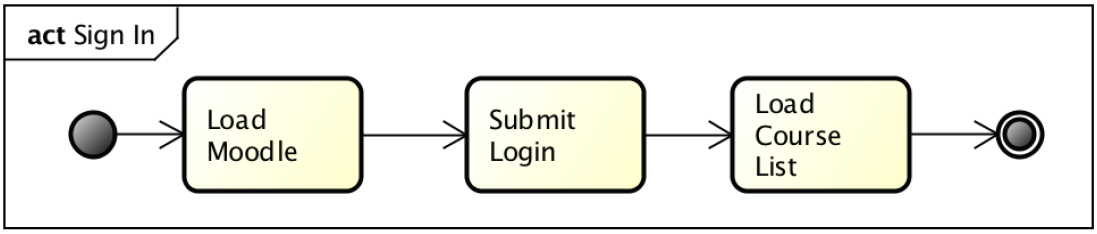


Figure D.2: UML activity diagram of the `Sign In` activity

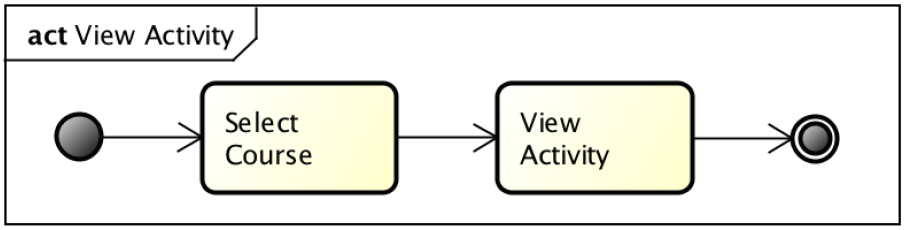


Figure D.3: UML activity diagram of the `View Activity` activity

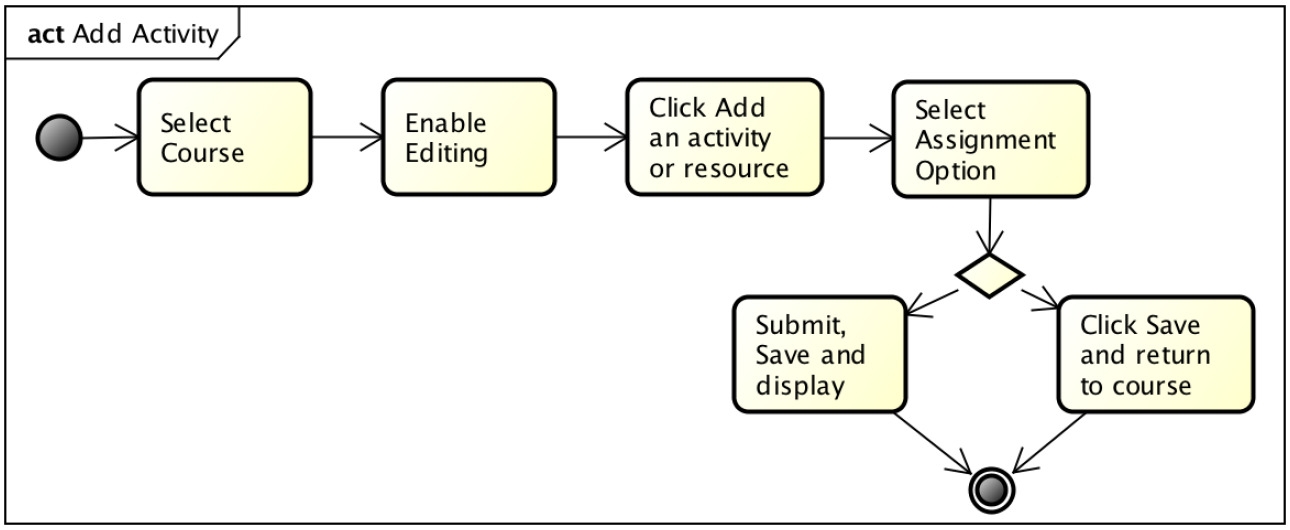


Figure D.4: UML activity diagram of the `Add Activity` activity

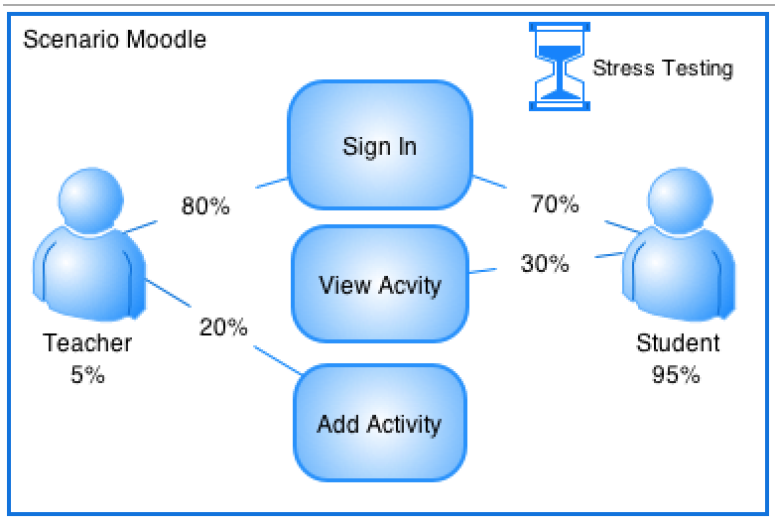


Figure D.5: `Canopus Performance Scenario` of the Moodle

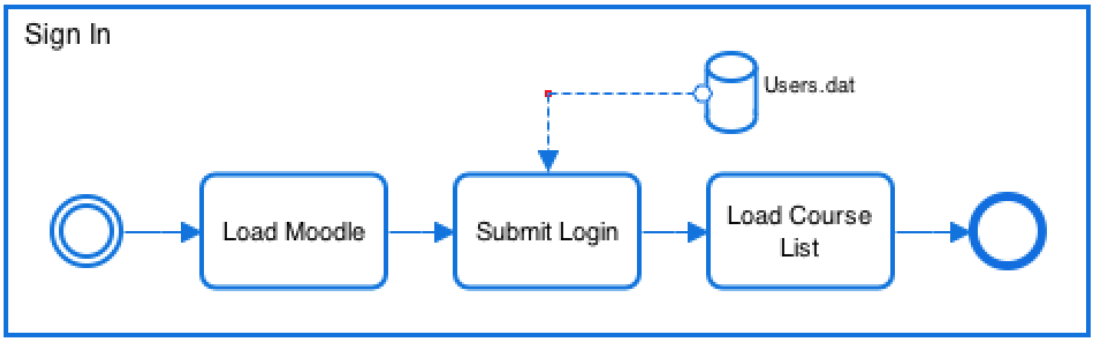


Figure D.6: `Canopus Performance Scripting` of the `Sign In` activity

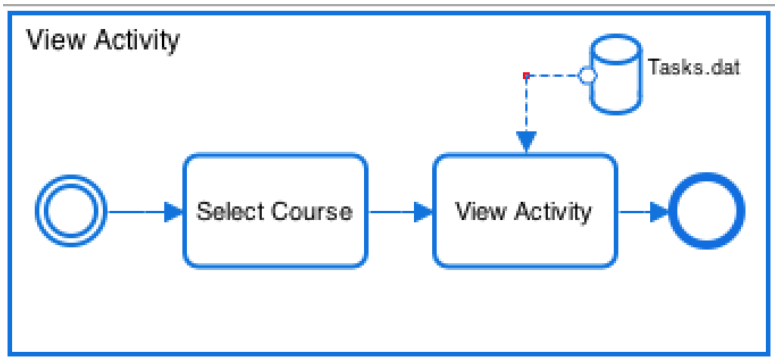


Figure D.7: Canopus Performance Scripting of the View Activity activity

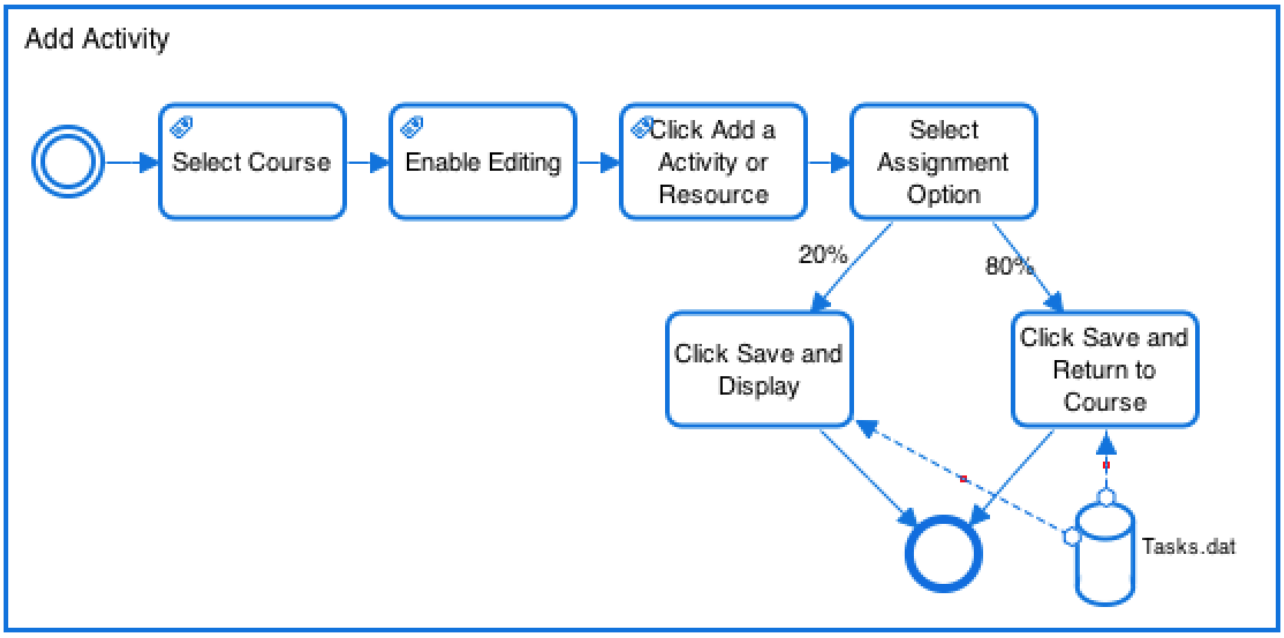


Figure D.8: Canopus Performance Scripting of the Add Activity activity

APPENDIX E – QUESTIONNAIRE PRE-EXPERIMENT

Questionnaire Pre-Experiment

Introduction

We would to participate in research study of DomainSpecific Language (DSL) for performance testing modeling. The study will provide an empirical evaluation of our DSL.

Please read this form and ask any question you may have before agreeing to participate in this study.

This study is being conducted by: Maicon Bernardino, Doctoral student at Pontifical Catholic University of Rio Grande do Sul (PUCRS), advised by Avelino Francisco Zorzo with collaboration of Elder Macedo Rodrigues, doctorate in Computer Science at PUCRS.

Background Information: The purpose of this survey is to evaluate the respondent's profile to map and randomize the experiment subjects between the experiment treatments.

Procedures: If you agree to participate in this study, we ask you to answer questions about your knowledge and skills on technical issues. The questions are multiple choice. It will take 5-10 minutes to complete the survey.

Risks: Being a participant in this study has no foreseeable risks.

Benefits: The researcher hopes to evaluate a DSL for performance testing modeling comparing to UML-based approach. It may help performance practitioners such as more efficient models of performance scenarios, and scripts. As a subject, you will have access to the results of this research.

Confidentiality: The records of this study and the secured files will be kept private. In any sort of report we might publish them not including any information which will make it possible to identify a subject. Only the primary researcher will have access to the records.

Contact and Questions: The researcher conducting this study is Maicon Bernardino. Please contact him with any question:

bernardino@acm.org

If you wish to participate, please start the survey by clicking in the next step.

Respondent's Profile

SQ1 Enter your name**. ***
Please write you answer here:

SQ2 Enter the type of higher institution where you study. *
Please choose **only one** of the following:

- ○ Private
- ○ Public

SQ3 Enter you course**. ***
Please choose **only one** of the following:

- ○ Computer Science
- ○ Computer Engineering
- ○ Information Systems
- ○ Software Engineering
- ○ System Analysis and Development
- ○ Other

SQ4 How many years of experience do you have on software engineering? *
Please choose **only one** of the following**:**

- ○ 14+
- ○ 11 - 13
- ○ 8 - 10
- ○ 5 - 7
- ○ 2 - 4
- ○ 0 - 1 year

Technical Profile

SQ5 How do you rate your technical knowledge in software modeling with UML? *
Please choose **only one** of the following**:**

- ○ Low, no prior knowledge
- ○ Regular, read a book or followed a course
- ○ Average, some industrial experience (less than 6 months)
- ○ High, industrial experience

SQ6 How do you rate your technical knowledge in Domain-Specific Language (DSL)? *
Please choose **only one** of the following**:**

- ○ Low, no prior knowledge
- ○ Regular, read a book or followed a course
- ○ Average, some industrial experience (less than 6 months)
- ○ High, industrial experience

SQ7 How do you rate your technical knowledge in performance testing? *
Please choose **only one** of the following**:**

- ○ Low, no prior knowledge
- ○ Regular, read a book or followed a course
- ○ Average, some industrial experience (less than 6 months)
- ○ High, industrial experience

SQ8 How do you rate your technical knowledge in modeling performance testing with notations or modeling languages? *
Please choose **only one** of the following**:**

- ○ Low, no prior knowledge
- ○ Regular, read a book or followed a course
- ○ Average, some industrial experience (less than 6 months)
- ○ High, industrial experience

SQ9 How do you rate your technical knowledge in modeling performance testing with UML? *
Please choose **only one** of the following**:**

- ○ Low, no prior knowledge
- ○ Regular, read a book or followed a course
- ○ Average, some industrial experience (less than 6 months)
- ○ High, industrial experience

Authorization

SQ10 I agree to participate in this experiment on the conditions that were proposed. I guarantee that I will perform this experiment the best way I can, ensuring that all information included here is real. *
Please choose **only one** of the following**:**

- ○ Agree
- ○ Disagree

Than you for completing our survey

Thanks for completing our brief survey and submitting you valuable information. Your answers will help to improve our domain-specific language and better tailor the performance testing modeling for the performance testing community.

APPENDIX F – QUESTIONNAIRE POST-EXPERIMENT

Questionnaire Post-Experiment

Introduction

We would to participate in research study of DomainSpecific Language (DSL) for performance testing modeling. The study will provide an empirical evaluation of our DSL.

Please read this form and ask any question you may have before agreeing to participate in this study.

This study is being conducted by: Maicon Bernardino, Doctoral student at Pontifical Catholic University of Rio Grande do Sul (PUCRS), advised by Avelino Francisco Zorzo with collaboration of Elder Macedo Rodrigues, doctorate in Computer Science at PUCRS.

Background Information: The purpose of this survey is to evaluate the respondent's profile to map and randomize the experiment subjects between the experiment treatments.

Procedures: If you agree to participate in this study, we ask you to answer questions about your knowledge and skills on technical issues. The questions are multiple choice. It will take 5-10 minutes to complete the survey.

Risks: Being a participant in this study has no foreseeable risks.

Benefits: The researcher hopes to evaluate a DSL for performance testing modeling comparing to UML-based approach. It may help performance practitioners such as more efficient models of performance scenarios, and scripts. As a subject, you will have access to the results of this research.

Confidentiality: The records of this study and the secured files will be kept private. In any sort of report we might publish them not including any information which will make it possible to identify a subject. Only the primary researcher will have access to the records.

Contact and Questions: The researcher conducting this study is Maicon Bernardino. Please contact him with any question:

bernardino@acm.org

If you wish to participate, please start the survey by clicking in the next step.

Experiment Questions

SQ1 When modeling with UML, the language provided all the elements required to model performance test scripts and scenarios. *****

Please choose **only one** of the following:

- ○ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

SQ2 The performance test models designed using the DSL approach express more adequately the functional and non-functional requirements than those models designed using the UML approach. *

Please choose **only one** of the following:

- ○ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

SQ3 It easier to design the performance test models applying the DSL approach than applying the UML approach. *

Please choose **only one** of the following:

- ○ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

SQ4 It more intuitive in the performance testing domain, to apply the DSL approach than to apply the UML approach. *

Please choose **only one** of the following:

- ○ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

Essay Questions

SQ5 Describe below what were the positive points identified during the design the performance testing models when applying the **UML approach**. *

Please write you answer here**:**

SQ6 Describe below what were the negative points identified during the design the performance testing models when applying the **UML approach**. *

Please write you answer here**:**

SQ7 Describe below what were the positive points identified during the design the performance testing models when applying the **DSL approach**. *

Please write you answer here**:**

SQ8 Describe below what were the negative points identified during the design the performance testing models when applying the **DSL approach**. *

Please write you answer here**:**

SQ9 Would you recommended the DSL approach for modeling performance testing to your peers or persuade your management to invest? If not why? If yes, what arguments would you use? *

Please choose **only one** of the following**:**

- ○ Agree
- ○ Disagree

Make a comment on your choice here**:**

Identification

SQ10 Enter your name**. ***

Please write you answer here:

Than you for completing our survey

Thanks for completing our brief survey and submitting you valuable information. Your answers will help to improve our domain-specific language and better tailor the performance testing modeling for the performance testing community.