UNIVERSIDADE FEDERAL DO PAMPA

João Batista Pedroso Carbonell

OpenMLPerf - Graphical Domain-Specific Modeling Language for Performance Testing in Web Systems

Alegrete
2023

João Batista Pedroso Carbonell

OpenMLPerf - Graphical Domain-Specific Modeling Language for Performance Testing in Web Systems

Master Thesis Master thesis presented as partial requirement for obtaining the degree of Masters of Software Engineering at Universidade Federal do Pampa.

Supervisor: Prof. Dr. Élder de Macedo Rodrigues

Co-supervisor: Prof. Dr. Maicon Bernardino da Silveira

Alegrete
2023

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

```
C264o  Carbonell, João
          OpenMLPerf - Graphical Domain-Specific Modeling Language
       for Performance Testing in Web Systems / João Carbonell.
          103 p.

          Dissertação(Mestrado)-- Universidade Federal do Pampa,
       MESTRADO EM ENGENHARIA DE SOFTWARE, 2023.
          "Orientação: Elder Rodrigues".

          1. Engenharia de Software. 2. Linguagem de Dominio
       Específico. 3. Teste de Desempenho. I. Título.
```

JOAO BATISTA PEDROSO CARBONEL

OpenMLPerf - Graphical Domain-Specific Modeling Language for Performance Testing in Web Systems

Dissertação apresentada ao Programa de pós-graduação em Engenharia de Software da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Mestre em Engenharia de Software.

Dissertação defendida e aprovada em: 11 de outubro de 2023.

Banca examinadora:

Prof. Dr. Elder de Macedo Rodrigues
Orientador
UNIPAMPA

Prof. Dr. Maicon Bernardino da Silveira
Co-orientador
UNIPAMPA

Prof. Dr. Fábio Paulo Basso
UNIPAMPA

Prof. Dr. Avelino Francisco Zorzo

PUCRS

Assinado eletronicamente por **ELDER DE MACEDO RODRIGUES**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 28/11/2023, às 20:17, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **MAICON BERNARDINO DA SILVEIRA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 28/11/2023, às 21:44, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **FABIO PAULO BASSO**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 01/12/2023, às 14:04, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Avelino Francisco Zorzo**, **Usuário Externo**, em 06/03/2024, às 15:43, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

“Science is a way of thinking much more than it is a body of knowledge.”
(Carl Sagan)

ABSTRACT

Testing system performance is one of the most crucial tasks in the software development process and its evolution. Most of the current performance tests focus on evaluating an already implemented system, making it difficult to predict issues such as load capacities and bottlenecks. Therefore, it is recommended to employ mechanisms for modeling and specifying system information before implementation. One potential solution is the utilization of models that represent unchangeable aspects of the system domain, such as infrastructure features or user behavior.

Using models to abstract, define, and model domain aspects has become the most commonly adopted approach for addressing problems within specific contexts, aligning with the principles of Model-Driven Engineering (MDE), which advocate for using models throughout the software lifecycle. Domain-Specific Languages (DSLs) are one way to specify and model a domain. DSLs are compact languages with limited expressiveness designed for specific domains. Unlike general-purpose languages like Python and Java, DSLs are not intended for implementing systems across all domains; they are restricted to their original purpose.

In this study, we propose the reimplementation of a graphical DSL called Canopus for modeling performance tests in web systems. As part of this reimplementation, we have developed a code generator within the DSL. The code generator is specifically designed to generate SCALA code that can be used in conjunction with the JMeter tool.

The previous implementation of Canopus utilized commercial licensed technologies, which limited its distribution and evolution. Hence, we present a new version of this DSL, named OpenMLPerf, which will be released under an open-source license, enabling its use, improvement, and evolution. With the addition of the code generator, OpenMLPerf empowers users to automatically generate SCALA code for their performance test scenarios, facilitating the integration with the widely-used JMeter tool.

For the selection of a DSL development tool, we conducted a Systematic Literature Mapping, identifying fifty-three (53) available DSL support tools. We chose to use Ecore from the Eclipse Modeling Framework (EMF) and the Sirius framework to implement the language metamodel and the language itself. In addition to these two frameworks, we also utilized the Acceleo framework for template modeling and Scala code generation for performance testing tools.

We conducted an empirical evaluation to measure the effort required by users when modeling a performance test scenario using the OpenMLPerf approach, compared to using a UML profile for performance testing. Although the results indicated that both approaches require similar efforts from participants, the OpenMLPerf approach proved to be highly scalable, comprehensive, and intuitive.

A crucial factor for adopting the OpenMLPerf approach is its development within an open-source platform, allowing for replication, customization, and community-driven improvements. As a point of consideration, it is worth noting that the adoption of this approach may be influenced by performance and usability issues related to the Eclipse platform.

Key-words: Domain-Specific Language. Model Based Testing. Model Driven Engineering. Performance Testing.

RESUMO

Testar o desempenho de um sistema é uma das tarefas mais cruciais no processo de desenvolvimento de software e sua evolução. A maioria dos testes de desempenho atuais concentra-se em avaliar um sistema já implementado, tornando difícil prever problemas como capacidades de carga e gargalos. Portanto, é recomendado utilizar mecanismos para modelar e especificar informações do sistema antes da implementação. Uma solução potencial é a utilização de modelos que representem aspectos imutáveis do domínio do sistema, como características de infraestrutura ou comportamento do usuário.

O uso de modelos para abstrair, definir e modelar aspectos do domínio tornou-se a abordagem mais comumente adotada para resolver problemas em contextos específicos, alinhando-se aos princípios da Engenharia Orientada a Modelos (MDE), que defendem o uso de modelos ao longo do ciclo de vida do software. Linguagens Específicas de Domínio (DSLs) são uma maneira de especificar e modelar um domínio. As DSLs são linguagens compactas com expressividade limitada projetadas para domínios específicos. Ao contrário de linguagens de propósito geral como Python e Java, as DSLs não se destinam a implementar sistemas em todos os domínios; elas são restritas ao seu propósito original.

Neste estudo, propomos a reimplementação de uma DSL gráfica chamada Canopus para modelar testes de desempenho em sistemas web. Como parte dessa reimplementação, desenvolvemos um gerador de código dentro da DSL. O gerador de código foi especialmente projetado para gerar código SCALA que pode ser usado em conjunto com a ferramenta JMeter.

A implementação anterior do Canopus utilizava tecnologias licenciadas comercialmente, o que limitava sua distribuição e evolução. Portanto, apresentamos uma nova versão desta DSL, chamada OpenMLPerf, que será disponibilizada sob licença de código aberto, permitindo seu uso, aprimoramento e evolução. Com a adição do gerador de código, o OpenMLPerf capacita os usuários a gerar automaticamente código SCALA para seus cenários de teste de desempenho, facilitando a integração com a amplamente utilizada ferramenta JMeter.

Para a seleção de uma ferramenta de desenvolvimento de DSL, conduzimos um Mapeamento Sistemático da Literatura, identificando cinquenta e três (53) ferramentas de apoio a DSL disponíveis. Optamos por utilizar o Ecore, do Eclipse Modeling Framework (EMF), e o framework Sirius para implementar o metamodelo da linguagem e a própria linguagem de modelagem. Além desses dois frameworks, também utilizamos o framework Acceleo para a modelagem de templates e a geração de código Scala destinado a ferramentas de teste de desempenho.

Realizamos uma avaliação empírica para medir o esforço exigido dos usuários ao modelar um cenário de teste de desempenho usando a abordagem OpenMLPerf, em comparação com o uso de um perfil UML para testes de desempenho. Embora os resultados tenham indicado que ambas as abordagens requerem esforços semelhantes por parte dos

participantes, a abordagem OpenMLPerf demonstrou ser altamente escalável, completa e intuitiva.
Um fator determinante para a adoção da abordagem OpenMLPerf é o seu desenvolvimento em uma plataforma de código aberto, o que permite sua replicação, customização e aprimoramento pela comunidade. Como ponto de atenção, vale ressaltar que a adoção dessa abordagem pode ser afetada por questões de desempenho e usabilidade relacionadas à plataforma Eclipse.

Palavras-chave: Linguagem Específica de Domínio. Teses Baseados em Modelos. Engenharia Dirigida por Modelos. Teste de Desempenho.

LIST OF FIGURES

Figure 1 – Process adopted in the development of the work 24
Figure 2 – Metamodeling architecture based on OMG's MOF. 30
Figure 3 – EMF unifies UML, Java, and XML 31
Figure 4 – Ecore hierarchical structure 32
Figure 5 – Sirius framework graphical interface 33
Figure 6 – Acceleo template-based technology 33
Figure 7 – Systematic Mapping Study Conduction Results 42
Figure 8 – Package dependency diagram showing the main model and the six language metamodels 49
Figure 9 – Class diagram of the OpenMLPerf package 50
Figure 10 – Class diagram illustrating the Performance Monitoring metamodel of the language 51
Figure 11 – Class diagram depicting the metamodel of the performance scenario in the language 52
Figure 12 – Class diagram displaying the scripting metamodel of the language . . . 54
Figure 13 – Sirius Specification Editor and the viewpoint specifications related to the models 55
Figure 14 – Modeled nodes to represent the elements of the *OpenMLPerf Specification Monitoring* metamodel 56
Figure 15 – Modeled nodes to represent the elements of the *OpenMLPerf Specification Metric* metamodel 57
Figure 16 – Example of graphical relationship elements (edges) between objects in the model. 57
Figure 17 – Monitoring Test Modeling Diagram 58
Figure 18 – Monitoring Metric Modeling Diagram 58
Figure 19 – Test Scenario Modeling Diagram 59
Figure 20 – Modeling diagram of a scenario script 59
Figure 21 – Workload modeling diagram 60
Figure 22 – Menu for creating the Scripting diagram 61
Figure 23 – Properties editing menu 61
Figure 24 – Results of the leveling questions 73
Figure 25 – Effort of participants 74
Figure 26 – Results of questions about OpenMLPerf 75
Figure 27 – Results regarding the recommendation of using the DSL 76
Figure 28 – Class diagram displaying the monitoring metamodel of the language . . 87
Figure 29 – Class diagram showing the metamodel of the workload 88
Figure 30 – Modeling diagram of the language's scripts 89

LIST OF TABLES

Table 1 – Number of participants per approach. 72
Table 2 – Effort of Participants (Average and Median) 73
Table 3 – Effort per participant 74
Table 4 – Standard deviation for participant effort 77

CONTENTS

1 INTRODUCTION . . . 21
1.1 Motivation . . . 21
1.2 Objective . . . 22
1.3 Methodology . . . 22
1.4 Organization . . . 25

2 BACKGROUND . . . 27
2.1 Model-Driven Engineering (Model-Driven Engineering (MDE)) 27
2.1.1 Model-Based Testing (Model-Based Testing (MBT)) . . . 27
2.1.1.1 Performance Testing (PT) . . . 28
2.1.2 Domain-Specific Languages . . . 29
2.1.2.1 Models, Metamodels, and Meta-Metamodels . . . 29
2.1.3 Eclipse Modeling Framework (EMF) . . . 30
2.1.3.1 Ecore . . . 31
2.1.4 Eclipse Sirius Framework . . . 31
2.1.5 Acceleo Code Generator . . . 32
2.1.6 DSL Canopus . . . 34
2.1.7 Lessons from the Chapter . . . 34

3 RELATED WORK . . . 35
3.1 Related Work . . . 35
3.1.1 Chapter Lessons . . . 37

4 SYSTEMATIC MAPPING STUDY ON DOMAIN-SPECIFIC LANGUAGE DEVELOPMENT TOOLS . . . 39
4.1 SMS Design and Execution . . . 39
4.2 Report . . . 42
4.3 Conclusion . . . 44

5 GRAPHICAL DOMAIN-SPECIFIC MODELING LANGUAGE FOR PERFORMANCE TESTING IN WEB SYSTEMS . . . 47
5.1 Requirements and Design Decisions . . . 47
5.1.1 Language Requirements . . . 47
5.1.2 Design Decisions . . . 48
5.2 Architecture and Domain Modeling with Eclipse Ecore . . . 48
5.3 Language Implementation with the Eclipse Sirius Framework 53
5.3.1 Definition of Relationships . . . 56
5.3.2 Modeling Diagrams . . . 57
5.3.3 Modeling Menus . . . 60

5.4 Code Generation 60
5.4.0.1 Acceleo Template Development 60
5.4.0.2 Integration with OpenMLPerf DSL 62
5.4.0.3 Customization and Extension 62
5.4.0.4 Generated Code Artifacts 63
5.4.0.5 Generation of Scala Code for JMeter 63
5.4.1 Chapter Lessons 66

6 EMPIRICAL EVALUATION 67
6.1 Evaluation Definition 67
6.1.1 Research Questions 67
6.1.2 Objective Definition 68
6.2 Planning 68
6.2.1 Context 68
6.2.2 Hypothesis Formulation 68
6.2.3 Participant Selection 69
6.2.4 Evaluation Design 69
6.2.5 Instrumentation 69
6.2.6 Threats to Validity 70
6.3 Evaluation Operation 71
6.3.1 Preparation 71
6.3.2 Execution 71
6.3.3 Results 73
6.3.4 Discussion 76
6.3.5 Chapter Lessons 77

7 CONCLUSION 79
7.1 Lessons Learned 79
7.2 Future Work 80

BIBLIOGRAPHY 81

APPENDIX 85

APPENDIX A – APÊNDICE A 87

APPENDIX B – 89

APPENDIX C – 91

APPENDIX D – **97**

cutter=M1234x, keywords=Modelo de texto, UNIPAMPA, Latex, firstcommitteemember=Nome membro da banca 1
UNIPAMPA, secondcommitteemember=Nome membro da banca 2
Instituição,

1 INTRODUCTION

The increasing prevalence of web applications hosted on local or cloud servers, it has become essential to test, predict, and analyze the capabilities and performance of the hosting infrastructure. Performance testing should be applied not only to systems hosted on physical or virtual machines but also to those hosted in the cloud or distributed in a hybrid manner. Executing and analyzing performance tests on an already implemented system helps verify whether resource utilization is efficient, plan architectural improvements, and prevent wastage of computational resources (MOLYNEAUX, 2014). Therefore, conducting tests to forecast future infrastructure demands aids in devising an efficient plan, encompassing not only the computational capacities of the infrastructure but also the technologies employed for system implementation.

One of the challenges in applying performance testing lies in integrating domain knowledge with the understanding of the hardware and software technologies used to create the hosting environment. One commonly employed approach to address this challenge is the utilization of the methodology of Model-Driven Engineering (MDE) (Model-Driven Engineering) and its associated activities, such as Model-Based Testing (MBT), which emphasize the use of models to express the domain and model the system responsible for solving the domain problem. MDE and MBT provide support for various activities that would typically be performed by the end user, such as code and script generation. This way, the user's efforts are concentrated solely on domain modeling. One such activity supporting this methodology is the utilization of Domain-Specific Languages (Domain-Specific Language (DSLs)).

DSL (FOWLER, 2010) has been widely adopted as an approach to solving problems in specific domains. It finds application in diverse domains, ranging from performance testing modeling (BERNARDINO, 2016), machine learning (ZHAO; HUANG, 2018), and domain-driven object-oriented design (LE; DANG; NGUYEN, 2018), to automatic generation of school timetables (RIBIĆ et al., 2018), and even languages for communication and querying of databases, such as Structured Query Language (SQL).

1.1 Motivation

Modeling tests without the concern for implementation platform technologies (FRANCE; RUMPE, 2007) is one of the benefits provided by DSLs for MBT. Bernardino (2016) proposed the Canopus DSL for modeling performance test scenarios and scripts for web systems. In addition to modeling test scenarios and scripts, Canopus aims to automatically generate input data for load testing tools such as *HP LoadRunner* (Hewlett Packard, 2018). Canopus development was a joint effort between industry and academia, through a partnership between researchers and a multinational Information Technology company.

The development of Canopus utilized a Lightweight Language (Language Workbench (LW)) called *MetaEdit+* (MetaCase, 2018). Although it was considered a satisfactory experience from a functional perspective, the fact that *MetaEdit+* is a commercially licensed distributed tool requiring each Canopus user to purchase a *MetaEdit+* license made the deployment and utilization of Canopus financially costly and, in many scenarios, unfeasible. Therefore, it became relevant to develop a new version that could be distributed under an open-source license, accessible for both industrial use and research and development in academia.

1.2 Objective

The main objective of this work is to develop an open-source graphical DSL for modeling performance test scenarios for web systems. This DSL will also include a code generation feature to automatically generate code snippets for input data in load generation tools, such as *JMeter* (Apache Foundation, 2023).

To achieve the main objective, the following specific objectives have been defined through requirements gathering and decisions (see Section 5.1):

- Implement a DSL that encompasses the following general requirements:
  - Open-source license
  - Enable graphical modeling and visualization of models
  - Represent the characteristics of the Performance Test (PT)
  - Model different metrics and scripts that can be reused by other scenarios
  - Model different user profiles and their behaviors
- Include a code generation feature to automatically generate code snippets for load generation tools like *JMeter*
- Evaluate the developed DSL through an empirical experiment

1.3 Methodology

To identify the existing tools, notations (graphical or textual), and their licenses, a Systematic Literature Mapping was conducted. The most important databases in the field of computer science were searched, applying inclusion, exclusion, and quality criteria. This allowed for the qualification and classification of the most relevant studies regarding DSL development tools in the context of this work.

As shown in Figure 1, the development process of the OpenMLPerf DSL was divided into:

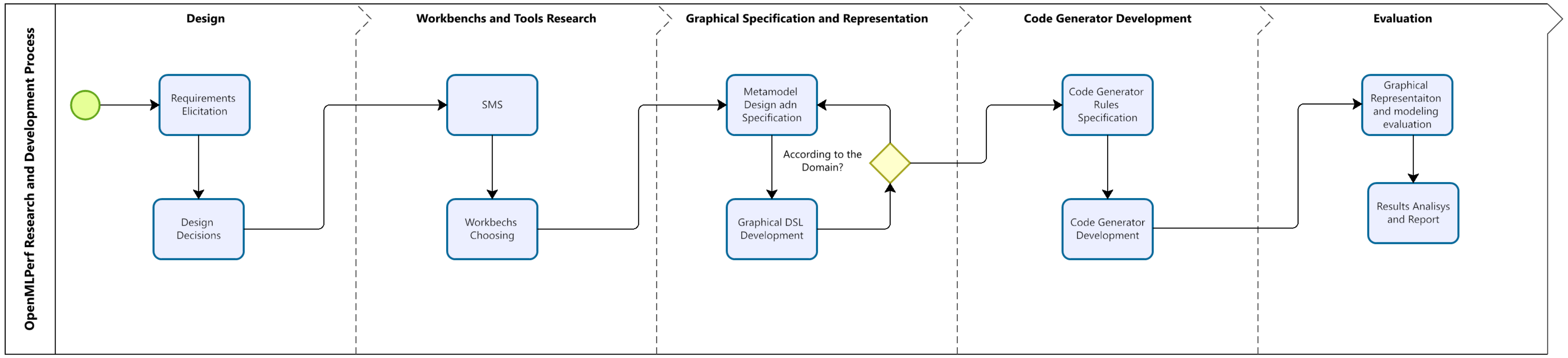
- Conception
  - Requirement elicitation for the language

    * Identification of the requirements necessary to meet the needs of performance testing domain.
  - Design decisions
    * Decisions necessary to fulfill the DSL requirements, such as the choice of implementation tools, code generation support, etc.
  - Systematic Literature Mapping
    * A systematic study to identify existing tools that support the creation and maintenance of DSLs.

⋆ Implementation
  - Modeling the metamodel using the Ecore framework
    * Definition of the properties, relationships, and elements of the domain and their representations in the metamodel.
  - Implementation of the visual representation of the language using the Sirius framework
    * Graphical representation of the elements defined in the model, serving as the language interface.
  - Code generation
    * Development of code generation mechanisms to automatically generate code based on the DSL models, facilitating their practical application in performance testing scenarios.

⋆ Evaluation
  - Empirical evaluation to assess the effort, effectiveness, and ease of use of the language.

Figure 1 – Process adopted in the development of the work


OpenMLPerf Research and Development Process
Design
Workbenchs and Tools Research
Graphical Specification and Representation
Code Generator Development
Evaluation
Requirements Elicitation
Design Decisions
SMS
Workbechs Choosing
Metamodel Design adn Specification
Graphical DSL Development
According to the Domain?
Code Generator Rules Specification
Code Generator Development
Graphical Representaiton and modeling evaluation
Results Analisys and Report

Source: Author

1.4 Organization

This work is organized as follows: Chapter 2 establishes the theoretical foundation to provide technical and theoretical background on the subjects and terms addressed throughout the work. Chapter 3 discusses related works that use DSL in some way for performance testing. Chapter 4 presents the Systematic Literature Mapping and its results, conducted to aid in finding the supporting tool used in the development of the proposed language. Chapter 5 presents the requirements and design decisions, the defined metamodel for the language, and the language implementation. Chapter 6 presents the empirical evaluation conducted to assess the *DSL*. Finally, in Chapter 7, the conclusions and future work will be discussed.

2 BACKGROUND

The concept of using models to express a domain and create software has been widely adopted in recent decades. The use of models in all phases of the software life cycle is a fundamental principle of Model-Driven Engineering (MDE). In this context, it is suitable to adopt models for abstracting test scenarios and generating code. Domain-Specific Languages (DSLs) enable domain modeling and subsequent code generation, which can be used as input for testing tools. In this section, we will discuss and explain terms that will be encountered and used in the following chapters and sections of this work, all of which are part of the domain of DSLs.

2.1 Model-Driven Engineering (MDE)

Model-Driven Engineering (MDE) is an approach in software engineering that utilizes models and their transformations to abstract various aspects of a domain and/or a system. According to (FRANCE; RUMPE, 2007), one of the primary goals of MDE is to isolate developers from the complexities of implementation platforms. In other words, it aims to hide the intricacies that occur during system execution, such as infrastructure, middleware, libraries, computer networks, etc. (KENT, 2002) states that MDE combines architecture, process, and analysis for domain abstraction. The activities of MDE include Model-Driven Architecture (MDA) and MBT.

2.1.1 Model-Based Testing (MBT)

Model-Based Testing (MBT) is an approach that utilizes models and their transformations to abstract various aspects of a domain and/or a system. According to (UTTING; LEGEARD, 2010), there are four main approaches to MBT:

The first approach is the generation of test input data for a domain model, this involves creating test input data using a domain model. Following we have the generation of test cases from an environment model and in this approach, test cases are generated based on an environment model. Other approach is the generation of test cases with Oracles from a behavioral model. In this approach, test cases, along with oracles for expected outcomes, are generated from a behavioral model. The last approach is the generation of abstract test scripts, where abstract test scripts are generated to represent test scenarios.

According to (EL-FAR; WHITTAKER, 2002), in addition to these approaches, several activities are typically carried out in MBT: Generation of Expected Inputs: Utilizing the model to generate test inputs, including test cases and test scripts; Generation of Expected Outputs: Creating mechanisms to determine if the test execution produces correct results; Test Execution: Running the test scripts and recording the results of each test; Result Comparison: Comparing the obtained results with the expected outputs to identify discrepancies; Decision-Making: Taking actions such as estimating the software's quality based on the test results; The last activity is the test completion, which conclues the testing process and preparing the software for release.

To carry out these tasks, supporting tools need to be adopted, allowing the full utilization of the advantages provided by MBT. Various tools for MBT support are available, including commercial, academic, and open-source tools (All4Tec, 2019)(Austrian Institute of Technology, 2019)(Test Optimal, 2019).

2.1.1.1 Performance Testing (PT)

One of the most important activities in Performance Engineeringis is the PT (SMITH; WILLIAMS, 1993), which focuses on improving scalability, and performance, and identifying bottlenecks in the tested systems. Performance testing is supported by measurement-based approaches, such as the analysis of non-functional requirements and performance metrics, e.g., response time, and throughput. Through performance testing, it is possible to determine the behavior of a system under different conditions, identifying processing limits, insufficient memory, and low disk space. According to Meier et al. (2007), performance testing can be classified into two categories:

- Load testing: It aims to determine or evaluate the behavior of the System Under Test (SUT) under normal workload conditions and determine if the system meets the requirements and specifications.
- Stress testing: It seeks to determine the behavior of the system under situations where the workload exceeds normal conditions, thus identifying system failures and bottlenecks.

Furthermore, according to Meier et al. (2007), the process of conducting performance tests consists of a set of activities:

Identify the Test Environment: Start by defining the test environment, including hardware specifications, software configurations, and network settings that will be used during testing. Identify Performance Acceptance Criteria: Establish goals and assumptions regarding response time and the utilization of computational resources by the system. This helps define what is considered acceptable performance. Plan and Model the Tests: Identify test scenarios and the variability of user representations. This involves creating test cases that cover different usage situations of the system. Configure the Environment: Prepare the testing environment, including the tools and resources needed to conduct performance testing efficiently. Implement Test Modeling: Develop scripts and test scenarios that represent real-world situations the system will encounter in production. Execute the Tests: Implement performance monitoring and run the tests according to the defined scenarios. This involves observing how the system behaves under load. Analyze the Results and Redo the Tests: After executing the tests, analyze the obtained results. This may involve creating summaries and consolidating collected data. If necessary, repeat the tests to validate findings and ensure result consistency.

2.1.2 Domain-Specific Languages

Domain-Specific Languages (DSLs) are also known as *Little Languages*, *Small Languages*, Special-Purpose Languages, or Domain-Specific Modeling Languages (DSMLs). As defined by Mernik, Heering e Sloane (2005), DSLs are "programming languages for computers with limited expressiveness that focus on a specific domain" (MERNIK; HEERING; SLOANE, 2005). Meanwhile, DSMLs are domain-specific languages with a different syntax that uses graphical components to model the behavior of the system (KELLY; TOLVANEN, 2007). A DSL can be classified in two ways: Internal DSL or External DSL.

Internal DSL: A DSL can be classified as internal when it uses the structure of a host language. As defined by Fowler (2010), "an internal DSL is a specific way of using a general-purpose language... a script, in an internal DSL, is valid code in its general-purpose language" (FOWLER, 2010).

External DSL: An external DSL is a language independent of the host language of the application. In an external DSL, a script can be parsed by code in the application, using techniques such as syntactic and semantic analysis (e.g., Structured Query Language - SQL) (FOWLER, 2010).

Language Workbench (LW) Language Workbenches are tools used to develop DSLs. According to Wachsmuth, Konat e Visser (2014a), LWs provide "high-level mechanisms for implementing programming languages to make the development of new languages accessible" (WACHSMUTH; KONAT; VISSER, 2014a). They are environments that not only support the creation of new DSLs but also work together with other necessary tools to enable the efficient use of these DSLs by end users. LWs facilitate not only the definition of syntax parsers but also help create customized environments for the languages. Additionally, they allow the definition of abstract syntax, which is commonly used to obtain the desired language, which can be accessed in an integrated development environment (KORENKOV; LOGINOV; LAZDIN, 2015).

Frameworks are sets of concepts used in a specific domain to solve problems. According to Johnson (1997), "a framework is an application skeleton that is ready to be customized by a developer" (JOHNSON, 1997). Frameworks are "components," and an application can use multiple frameworks. However, frameworks can be adapted according to the user's needs, providing more flexibility than other components (JOHNSON, 1997).

2.1.2.1 Models, Metamodels, and Meta-Metamodels

According to Fuentes-Fernández e Vallecillo-Moreno (2004), the concept of organizing models, metamodels, and meta-metamodels based on the Meta-Object Facility

(MOF) of the OMG is classified into four layers.

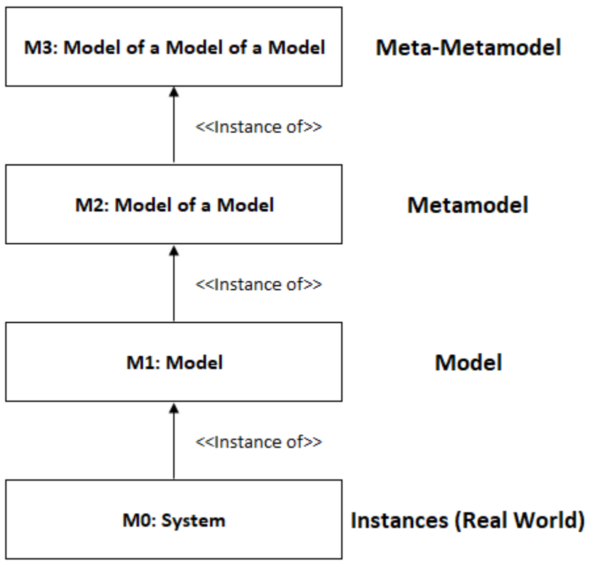
Layer 0 (M0): This layer is named "instance" and its function is to model the running system. Its elements are real instances that exist in the system. Examples of these instances are "Shneider" living in "Alabama" who bought a copy of the book "DSL".

Layer 1 (M1): A model is a description of a system (or part of it) written in a well-defined language. A well-defined language is a language with syntax and semantics suitable for automatic interpretation by a computer. Examples of this layer are "Person", "State", and "Book".

Layer 2 (M2): A model of a model (Metamodel) consists of elements that comprise the modeling language. This layer defines the concepts that will be used to model an element of the model layer, and each element of the model layer is an instance of an element in the metamodel. For example, in UML, "Class", "Attributes", and "Relationships" are defined in M2.

Layer 3 (M3): A model of a model of a model (Meta-Metamodel) defines the concepts that can be used to define modeling languages. For example, UML defines "Class" as a classifier.

Figure 2 – Metamodeling architecture based on OMG's MOF.



Tang (2009)

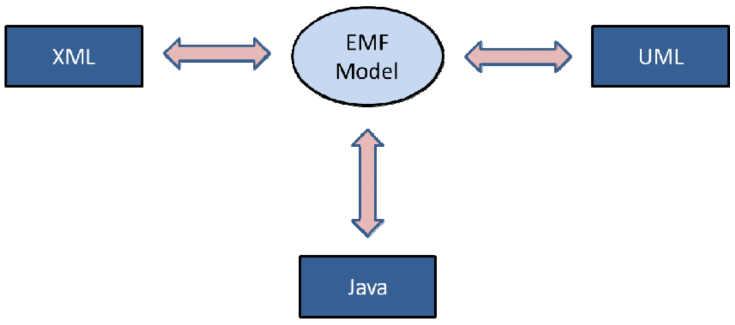
Frameworks and DSLs can be classified as metamodels following the specifications contained in OMG's MOF and DSL, however, they are classified as models.

2.1.3 Eclipse Modeling Framework (EMF)

Developed and maintained by the Eclipse Foundation, the EMF is a framework that enables the abstraction and modeling of domains through the definition of models.

The modeling can be done using UML modeling tools such as class diagrams or Java annotations, as well as XML schemas, allowing code generation from the defined models. According to Steinberg et al. (2008), EMF unifies three important technologies (see Figure 1): Java, XML, and UML. Regardless of the technology used to define the model, EMF brings them together under an EMF model that maintains their unity. The Eclipse EMF is distributed under the Eclipse Public License 2.0.

Figure 3 – EMF unifies UML, Java, and XML



Source: Steinberg et al. (2008)

2.1.3.1 Ecore

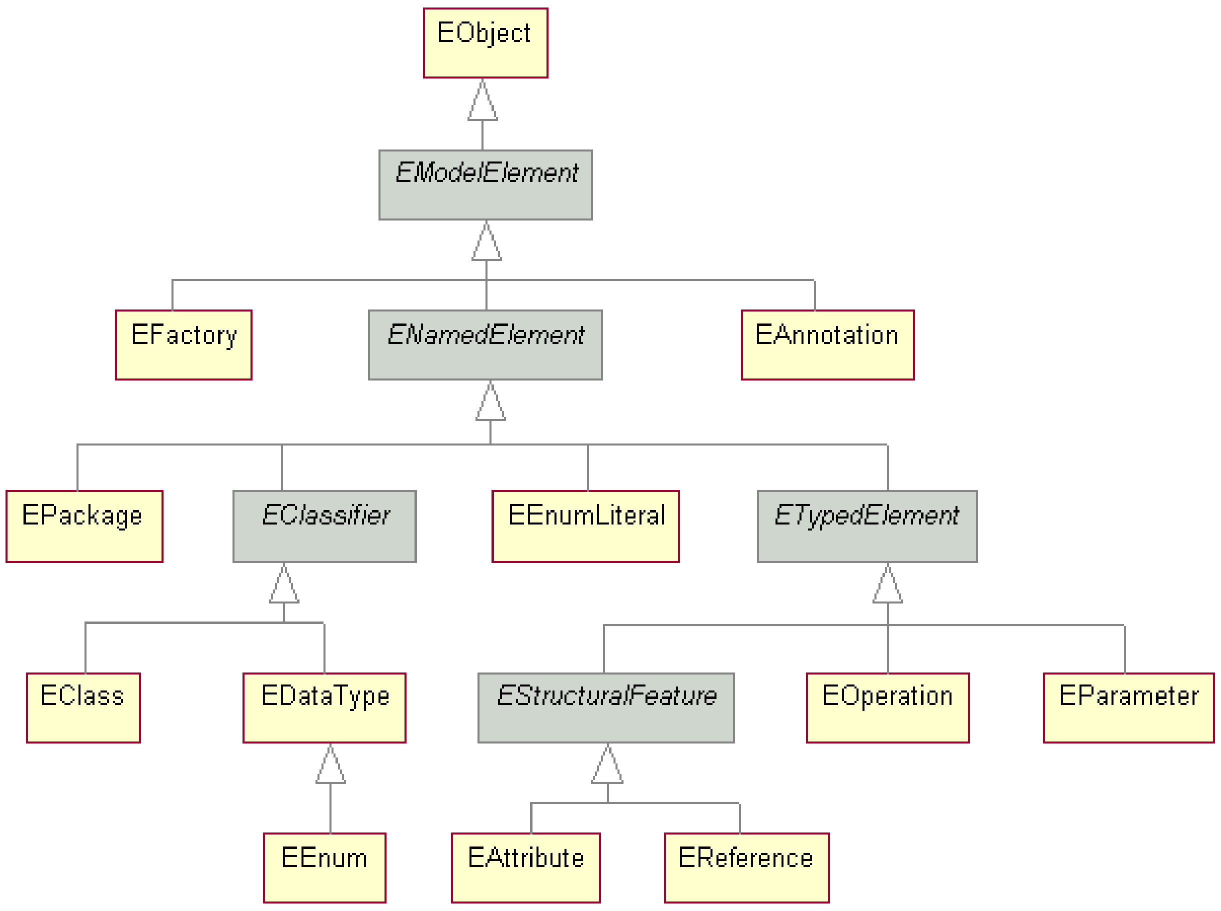
Ecore is the model used to represent other models in the Eclipse Modeling Framework (EMF). According to Budinsky et al. (BUDINSKY et al., 2004), Ecore is its own metamodel since Ecore is an EMF model itself.

The concept of Ecore model components is arranged following a hierarchy based on a UML dialect. The top-level component is called **EObject**, from which another abstract component called **EModelElement** is derived. At this point in the Ecore hierarchical tree, we find the **ENamedElement** component, from which all components below in the hierarchy inherit the definition of the ***name*** attribute, except for **EAnnotation** and **EFactory** components that are located alongside. Below this level, there are other components such as **EClassifier**, **ETypedElement**, **EClass**, **EDataType**, **EAttribute**, **EReference**, among others (see Figure 4).

2.1.4 Eclipse Sirius Framework

The Sirius framework is an open-source project owned and maintained by the Eclipse Foundation, with the goal of enabling the creation of modeling tools. Sirius encapsulates EMF, GEF, and GMF. EMF provides a graphical object in the form of a model for representation, GEF provides the capability of building visual representations of models, and GMF provides the tool for creating graphical editors Viyović, Maksimović e Perišić (2014). In Sirius, editors are defined by a model that establishes the behavior of all editing and navigation tools used to adapt a DSL.

Figure 4 – Ecore hierarchical structure



Source: Eclipse Foundation (2018a)

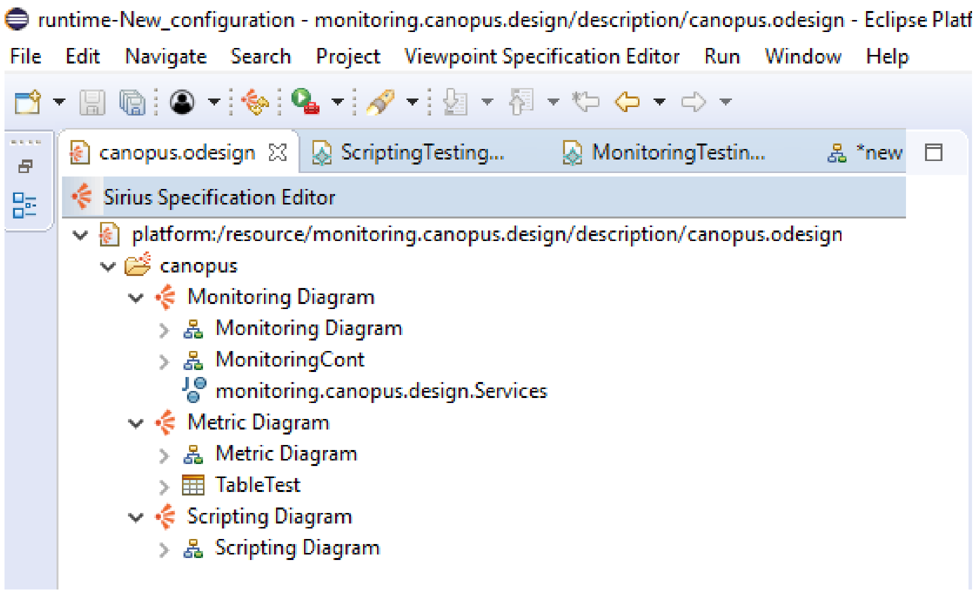
The construction of the modeling tool, or DSL, is done following the domain modeling definitions established in an Ecore model. It is possible to establish graphical representations for both relationships and elements (see Figure 3). It is also possible to choose parts of the model to be represented and modeled, as well as parts that have semantic meaning in the model but are hidden from the user, serving only as model rules.

2.1.5 Acceleo Code Generator

Acceleo (Eclipse Foundation, 2023) is a code generation framework widely used in model-driven development (MDD) approaches. It allows developers to define templates that specify how to transform models into code. With Acceleo, it is possible to generate source code, configuration files, documentation, and other artifacts from models expressed in various modeling languages.

One of the key features of Acceleo is its template-based approach. Developers create templates using a combination of static text and dynamic expressions written in a dedicated Acceleo language. These templates can access and manipulate the elements of

Figure 5 – Sirius framework graphical interface


runtime-New_configuration - monitoring.canopus.design/description/canopus.odesign - Eclipse Platf

File Edit Navigate Search Project Viewpoint Specification Editor Run Window Help

canopus.odesign | ScriptingTesting... | MonitoringTestin... | *new

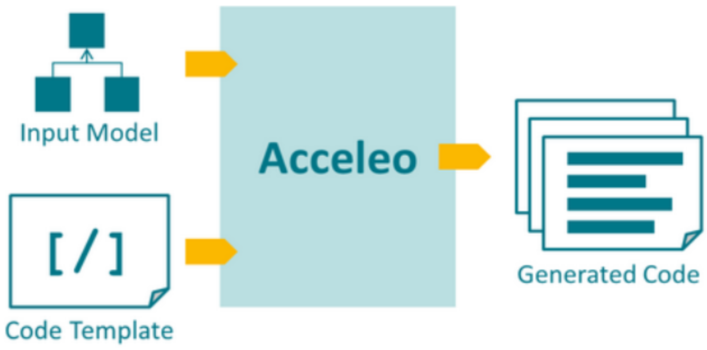
Sirius Specification Editor

- platform:/resource/monitoring.canopus.design/description/canopus.odesign
  - canopus
    - Monitoring Diagram
      - Monitoring Diagram
      - MonitoringCont
      - monitoring.canopus.design.Services
    - Metric Diagram
      - Metric Diagram
      - TableTest
    - Scripting Diagram
      - Scripting Diagram

Source: Author

a model, extracting relevant information and generating code accordingly. The generated code can be customized and tailored to meet specific requirements and constraints (see 6).

Figure 6 – Acceleo template-based technology



Source: (Eclipse Foundation, 2023)

Acceleo provides a set of tools and APIs to support the code generation process. It integrates with popular modeling tools and frameworks, such as Eclipse Modeling Framework (EMF) and Eclipse Modeling Project (EMP), making it a versatile choice for model-driven code generation tasks. It also supports the use of model transformations, allowing developers to define complex mapping rules between models and code.

The Acceleo code generator offers benefits such as improved productivity, consistency, and maintainability in software development projects. By automating the generation of code from models, it reduces manual coding efforts and minimizes the risk of errors and inconsistencies. It also promotes model-driven practices, enabling developers to focus on high-level modeling concepts and abstracting away low-level implementation details.

In the context of OpenMLPerf, the Acceleo code generator plays a crucial role in translating the models created with the OpenMLPerf DSL into executable performance testing artifacts. It takes the model representations of performance testing scenarios, metrics, workloads, and other elements and generates the corresponding code files that can be executed to conduct performance tests. This integration with Acceleo further enhances the usability and effectiveness of OpenMLPerf, enabling seamless transitions from modeling to execution and facilitating the automation of performance testing processes.

2.1.6 DSL Canopus

The Canopus (BERNARDINO; ZORZO; RODRIGUES, 2016) Domain-Specific Language DSL was created using the MetaEdit+ Language Workbench (MetaCase, 2018), a proprietary tool developed and maintained by MetaCase. The primary emphasis of the Canopus DSL lies in facilitating performance testing for web systems within a specific domain. Canopus allows the modeling of various aspects of the performance testing domain, such as defining user profiles, scalability of access and resources, and monitoring of the System Under Test (SUT) and its associated metrics. It also allows the modeling of scripts that describe the activities and behavior of the modeled user profiles. This information is associated with the modeling of scenarios in which these profiles and scripts will be executed.

2.1.7 Lessons from the Chapter

In this chapter, the main concepts involved in our work have been explored. Model-Driven Engineering (MDE), Model-Based Testing (MBT), and particularly Domain-Specific Languages (DSL) have been discussed to understand the impact and the role of our proposal in the software development process. The use of models to represent concepts and properties is widely adopted by both industry and academia. Therefore, it is important to focus on the use of these representations rather than the specific execution technologies. MBT naturally fits into this context as every system, even if not implemented yet, needs to be tested, and the best way to test a non-existent system is through models that represent domain aspects. Thus, discussing concepts such as model, metamodel, and meta-metamodel helps in understanding the implementation mechanisms of the language and its integration with the real world. Understanding these concepts not only assists in the implementation of our proposal but also in choosing the most suitable tool for its implementation.

3 RELATED WORK

Multiple research works address the realm of Domain-Specific Languages DSLsacross various domains (NAKAMURA et al., 2012), (PEREZ; VALDERAS; FONS, 2013), (RIBIĆ et al., 2018). It is known that, as of the date of this work, only the Canopus DSL (BERNARDINO, 2016) has been developed to perform graphical modeling of performance tests. In this chapter, some tools for performance testing will be presented and discussed, where some are DSLs themselves, while others use DSLs to assist the tool in its purpose.

3.1 Related Work

Ruffo et al. (RUFFO et al., 2004) in their work presents a tool for performance evaluation of web applications with a focus on user behavior. Called *WALTy (Web Application Load-based Testing tool)*, it consists of a set of tools that enable scalable test analysis. To achieve this goal, load tests are based on information extracted from application logs using Customer Behavioral Model Graphs (CBMG) (MARK; CSABA, 2007), which are proposed for characterizing state transition graphs of e-commerce websites. CBMGs are used to represent each section, and a clustering algorithm is then employed to group all identified sections. This allows the identification of user profiles for a group of users with common navigation patterns. The process of *WALTy* can be structured as follows: Firstly, CBMG constructors, using input data (logs), define states based on a set of user rules, specify embedded objects and where these objects should be filtered, and generate parameter specifications to characterize user website behavior, considering the number of clusters and session time limit. Subsequently, the network traffic representation is generated using a CBMG with the aid of an algorithm.

One of the key differences between this work and our proposed approach is the timing of the testing. While in the work of Ruffo et al. (RUFFO et al., 2004), the system needs to be fully implemented, in our approach, only the system concept is required, as we model the application domain and its concept, allowing for pre-modeling and testing before implementation. Furthermore, our approach is visual, making it more technology-agnostic in terms of the adopted execution technologies.

In the work of Abbors et al. (ABBORS et al., 2012), the model-based performance testing tool MBPeT is presented. The tool uses models to generate workload to be applied in real-time to systems, collecting different performance metrics. The models are defined using timed probabilistic automata, describing different user profiles that interact with the system. As input, the tool accepts a set of models expressed as automata, the number of virtual users, the ramp-up functions, and the test duration. It then provides a report describing the test measurements.

The performance models consist of user behavior, described by locations and transactions. The models are read by parsers, which build the model representations and validate them based on a set of rules, such as ensuring valid connections between locations.

The test is reported by a report generator module, which creates an HTML report with all the test execution results. The load generator sends a workload to the System Under Test (SUT), where each instance of a model contains user actions expressed in probabilities. Similar to the work of Ruffo et al. (RUFFO et al., 2004) mentioned earlier, the system needs to be defined and built prior to testing, and there is no graphical representation for all stages of the performance testing process, only for the automata models. In contrast, in our approach, the visual aspect encompasses all stages of performance testing modeling.

A declarative approach for performance testing in continuous software development environments is presented by Ferme et al. (FERME; PAUTASSO, 2018). The approach consists of a DSL for performance testing and a programmable framework driven by this DSL, targeting the end-to-end execution process of these tests. The tests can be specified using templates of standard test models or more advanced models. The DSL is implemented as an expressive goal-oriented language. The tests are specified declaratively, with automated and model-driven executions. The second part of the approach, related to the framework, uses the DSL itself to program and automate the test execution process, performing all the activities and objectives specified by the users for the test execution. The proposed DSL is used to declaratively specify the test objectives and control the execution processes, and it is adapted for microservice systems. Some of the performance characteristics that can be specified using the DSL include load functions, workloads, virtual users, test data, test set management, and performance data analysis. According to the authors, the framework was built to assist users in all activities of performance testing. The approach proposed by the authors presents a textual DSL with a declarative language, tightly bound to performance testing terms, which may make it less user-friendly for users in the proposed domain. Our approach also utilizes terms and properties of performance testing, but we consider it much more accessible to users in the domain of web applications. Additionally, being visual, our approach complements attribute and property information with graphical representations of the domain and relationships between elements.

Sun et al. (SUN; WHITE; EADE, 2014) present a textual DSL for specifying high-level load test plans and quality of service (QoS) requirements without details of low-level configurations. The DSL, called GROWL, is implemented using the Xtext framework. From the DSL, a model is constructed that generates test specifications compatible with the jMeter load testing tool. The DSL is used to capture resource configuration and information about QoS that are not captured by load-generation tools. The model analysis is used to identify the relationship between QoS performance and resource configurations, and code generation is performed for the automatic allocation of these configurations. Through an instance of a GROWL model, an XML-based specification is generated with the assistance of xTend.

GROWL satisfies the modeling, model analysis, and code generation parts within the ROAR (Resource Optimization, Allocation, and Recommendation System) approach proposed by the paper, which combines modeling, model analysis, test automation, code generation, and optimization techniques. In comparison to our work, the approach presented by Sun et al. (SUN; WHITE; EADE, 2014) does not allow for the modeling of complex behaviors through input parameters, i.e., variable parameters as test inputs. The parameters are specified in the modeling phase, whereas in our approach, the parameters can be dynamic and come from external files.

The work of Spafford e Vetter (2012) describes an approach for analytical performance modeling that utilizes a DSL called Aspen. Aspen specifies a formal grammar to describe two types of models: one that describes the behavior of an application, including parallelism, counters, data structures, and control flows, and another that specifies an abstract machine model. Both specified models are compiled, checked for semantic correctness, and persisted or simulated for modeling scenarios. In contrast to OpenMLPerf, which focuses on measurement models that can be compared with other techniques, Aspen considers predictive models to compare performance with other techniques, such as analytical models.

Some performance testing support tools have been developed in the past years. Gatling (Gatling, 2018) is a load-testing tool where scenarios are defined in code using an expressive DSL, making the scenarios self-explanatory. Load Runner by (Hewlett Packard, 2018) measures the behavior of a system under load by simulating user activities and generating messages between applications. Both solutions allow for scenario simulation and performance testing generation. However, unlike the proposal of this work, neither of them allows for graphical modeling of test scenarios independent of load and performance testing technologies.

3.1.1 Chapter Lessons

The use of DSLs to represent performance tests has gained increasing popularity when deciding which approach to use for modeling domain test models. There are both tools composed of a subset of other tools, including DSLs, and the use of specific-purpose languages alone. However, despite this variety of modeling tools, none of them, except for OpenMLPerf and Canopus DSL, have been found to allow for graphical modeling of performance tests. The majority of tools in this domain support textual notation for modeling.

In this chapter, we have presented related works to our proposed web system performance testing modeling tool. Some works make use of DSLs at some stage of the development, execution, or user utilization process. We have also introduced some performance testing execution tools that could potentially be future output tools for OpenMLPerf.

4 SYSTEMATIC MAPPING STUDY ON DOMAIN-SPECIFIC LANGUAGE DEVELOPMENT TOOLS

In this chapter, we present a detailed summary of a Systematic Mapping Study Systematic Mapping Study (SMS) on Domain-Specific Language (DSL) (IUNG et al., 2020) development tools that we conducted. The objective of the study was to identify various development tools for DSLs, including LWs, code generators, analyzers, and code transformers, among others, that could provide support at any stage of DSL development.

4.1 SMS Design and Execution

For this SMS, the review process presented by (PETERSEN et al., 2008) was adopted. The process was divided into three phases: Planning, Execution, and Reporting. The first two phases will be addressed in this section.

The first step was to define the objective of the study, which is to map LWs, frameworks, and other supporting tools, in order to provide an overview of tools and their licenses and assist DSL developers. This aims to help practitioners in MDE (Model-Driven Engineering) choose the best tools that can support them in their challenges and needs.

With this goal, the following Research Questions (RQ) was formulated:

RQ1. *What are the technologies used for DSL development?* Our goal is to characterize these tools, thus qualifying their maturity in research and practice.

RQ2. *What are the tools license types?* Once tool acquisition and training are dependent from decisions such as business models built on non-commercial or commercial licenses, our goal is to map tools by licensing.

RQ3. *For which application domains are the studies devoted to?* Our goal is to identify the relevance of DSL roles in software development for each cross-domain such as Web applications, mobile applications, embedded systems, and others.

RQ4. *What features of the DSL creation process do these tools support?* Since model management tools are diverse and help in many phases of DSL development, we aim at characterizing these proposals by their technical features.

To determines which works may be classified and qualified in later stages of the study, inclusion and exclusion criteria activity based on the scope, study objectives, and the research questions were defined, as described next:

Inclusion Criterias (Inclusion Criteria (ICs))

IC1. *The primary study must present an approach, technique, method, process, tool, framework, or LW to manipulate DSL or DSML.*

Studies presenting a new technology that support DSL or DSML development will be included, *e.g.* Studies presenting the Xtext framework.

IC2. *The primary study must mention a DSL supporting tool, framework, or LW.*

Studies that mention a technology that supports DSL or DSML development will be included, *e.g.* Studies that are focused on the geospatial domain but mention the use of a DSL and the technology that was used in its development.

Studies that focus on presenting a new DSL or DSML and also mention the technology that is used in its development will be included, *e.g.* Studies focused on presenting a DSL and that also mention the technology that was used in its development.

Exclusion Criterias (Exclusion Criteria (CEs))

CE1. *Studies published before 2012.*

CE2. *Studies in any language other than English.*

CE3. *Duplicated and/or incomplete studies.*

CE4. *Studies only available in the form of abstract, slide presentation, poster or short paper.*

During the search process, we only considered the use of computer science databases providing web-based search engines through keywords. The following digital libraries on the computing area are used: Compendex (Engineering Village)[1], IEEE Xplore[2], ScienceDirect[3], Association for Computing Machinery (ACM) Digital Library[4], Scopus[5] and SpringerLink[6].

Study Quality Assessment To evaluate the quality of the primary studies, a set of Quality Assessments (Quality Assessments (ICs)) criteria was defined. The studies target for IC is those remaining after application of the inclusion and exclusion criteria. These criteria aim to quantify the relevance of each primary study and to allow us to compare the selected primary studies. Primary studies with “zero (0) quality score” are removed from the relation of primary studies, even if they are in the domain of the research. In addition, primary studies that did not score in QA1 consequently did not score on other quality assessments and were therefore removed from the list of primary studies. Each researcher applied the IC criteria in accordance with the following grade: **Y**es: 1.0; **P**artially: 0.5; **N**o: 0.0. The IC criteria are defined as follows:

[1] *Compendex*: <www.engineeringvillage.com>
[2] *IEEE*: <www.ieeexplore.ieee.org>
[3] *ScienceDirect*: <www.sciencedirect.com>
[4] *ACM*: <https://dl.acm.org>
[5] *Scopus*: <www.scopus.com>
[6] *SpringerLink*: <www.link.springer.com>

IC1. *Does the study present a tool that supports DSL development?*
Evaluation: **Y**: The study presents a tool that supports DSL development; **P**: The study mentions a tool that supports DSL development but does not present details about the tool; **N**: The study does not present or mention a supporting tool to assist DSL development.

IC2. *Does the tool support at least one of the notations (graphical or textual)?*
Evaluation: **Y**: The study presents a tool supporting graphical or textual notations; **P**: The study does not present any supporting tools but the results indicate that graphical or textual notations were used; **N**: The study does not indicate evidences about a graphical or textual notation usage.

IC3. *Does the study report how the tool was applied in the development of a DSL?*
Evaluation: **Y**: The study presents the tool using coding examples, code snippets or images; **P**: The study argues that a tool was used, but does not present implementation details; **N**: The study does not present any implementation evidences.

IC4. *Is the tool usage described in a clear/detailed manner?*
Evaluation: **Y**: The study presents a tutorial, a process, steps to complete a task, or defines a flow to achieve a goal; **P**: The study only mentions activities that must be executed without further explanations; **N**: The study does not present any evidence in order to support the usage of the tool.

Data Extraction Strategy

We created a form to identify and extract relevant data from the selected studies. This information is used to answer the RQs. From each study, we extracted the following data:

- ⋆ Database: ACM, Compendex (Engineering Village), IEEE, SCOPUS, ScienceDirect and SpringerLink
- ⋆ Source: full reference conference, book, journal name
- ⋆ Title
- ⋆ Abstract
- ⋆ Authors
- ⋆ Year
- ⋆ Application Domain: domain to which the study is proposed
- ⋆ Tool, Framework, Language Workbench
  - – Feature: notation, semantics, edit support, semantic and syntactic services, validation, testing and composability
  - – Licence Type: commercial or non-commercial

Execution

Two search strings were used to search the IEEE Xplore digital library. This composite search strategy was adopted because the IEEE search engine has a limitation of 15 terms per string. Therefore, to accommodate the common terms, the search string was split into two compound terms: DSL and DSML.

For all databases, the search was restricted to the fields "Abstract," "Title," and "Keywords." Additionally, the search engine was configured to include papers published from 2012 to 2019.

As depicted in Figure 7, a total of 1,862 studies remained after removing duplicates. Subsequently, studies that were deemed irrelevant to the research scope, despite containing the terms DSL or DSML, were excluded. This step reduced the total to 1,780 studies, which were subjected to the application of exclusion and inclusion criteria.

The activities "Application of Exclusion Criteria" and "Application of Inclusion Criteria" were executed, resulting in 1,780 studies. These studies were then read and assessed in the "Qualifying and Classifying Papers" activity. As a result, 1,430 studies remained after applying the exclusion criteria. Next, studies that did not relate to any inclusion criteria were excluded, leaving a total of 390 studies for qualification and classification.

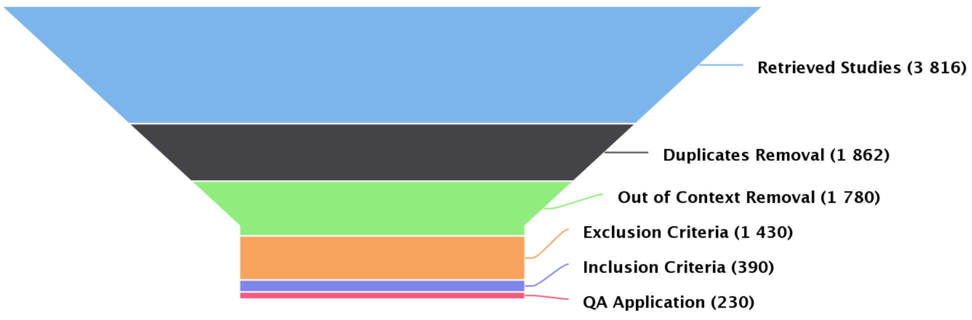


Figure 7 – Systematic Mapping Study Conduction Results

4.2 Report

In this section, we present the outcomes that address the research questions as follows.

RQ1. *Which are the technologies used for DSL development?*

After analyzing the 230 selected papers, we identified 59 DSL development tools. It was observed that some technologies were mentioned by multiple authors. Notably, the Xtext framework received a significant number of citations (102), along with other tools based on the Eclipse Modeling Framework (EMF) (114) and Graphical Modeling Framework (GMF) (30). Examples of such tools include the Papyrus tool (18) and the Sirius framework (42). Several other LWs also received notable citations, including MetaEdit+

(27) from MetaCase, MPS (Meta Programming System) (48) developed by JetBrains, and Spoofax (21) by MetaBorg.

Out of the mapped tools, 28 are graphical supporting tools, accounting for 47.5% of the total, making graphical notations the most widely used. On the other hand, 20 tools (34%) support textual notation.

Among the mapped tools, `T38` and `T55` provide support for all notations, accounting for 3.4% of the total. Only `T34` offers support for graphical, tabular, and symbolic notations. Additionally, `T32` supports graphical, textual, and tabular notations. Tools such as `T5`, `T6`, `T21`, `T28`, `T33`, `T43`, and `T48` provide support for both graphical and textual notations, representing 11.9% of the total number of tools.

It is important to note that none of the tools implement all notations, as some tools focus on specific activities within DSL development. Furthermore, although `T1`, `T27`, and `T58` are code generators, `T57` is a Java dialect, and `T18` is a family of languages and tools for code generation and model validation, they are considered part of the textual notation in this context.

RQ2. *What are the tools license types?*

In this SMS, the studies were classified into two categories: commercial or non-commercial. The information regarding the license type was typically extracted from the studies themselves, but in some cases, it was necessary to search official websites and manufacturers of the tools. In the analises was classified 41 tools as non-commercial and only nine having commercial licenses. The license type of nine tools could not be identified as they have not been officially released and do not have online information. To gather information about these tools, we conducted searches on Google (<www.google.com>). The category of non-commercial software includes tools under academic licenses and those available from online repositories open for evaluation, such as GitHub.

RQ3. *For which application domains is the study devoted?*

To map the domains in which the construction tools for DSLs are applied, we examined the sections dedicated to evaluation and conceptual demonstration in the selected studies. Based in the data collected from these studies, it can be concluded that "DSL construction" is a mature research area, with numerous cross-application domains, indicating increasing interest from the research community.

It is observed that 36.1% (83) of the included studies are related to tools, approaches, or methods that support various stages of the DSL and DSML development process lifecycle. Embedded systems are the focus of 3% (seven studies), while web systems account for 1.3% (three studies). Mobile applications are mentioned in 2.6% (six studies), and multi-agent systems are represented by 1.7% (four studies). Additionally, there are four studies specifically targeting Cyber-Physical Systems, making up 1.7%. Application domains mentioned in only one or two studies, such as aerospace systems, are classified as "Other," which comprises the majority of the studies at 53.5% (123 papers). This diverse range of application domains demonstrates the breadth of research in the field.

RQ4. *What features of the DSL creation process do these tools support?*

It is noteworthy to highlight the comprehensive capabilities of non-commercial LWs such as Xtext, MPS, GEMOC Studio, and MetaEdit+. These tools cover multiple features in the DSL development process. Xtext (Eclipse Foundation, 2018b) supports 29 features, MetaEdit+(MetaCase, 2018) covers 28 features, MPS(JetBrains, 2023) covers 32 features, and GEMOC Studio (Eclipse GEMOC Research Consortium, 2023) covers 31 features. Spoofax (WACHSMUTH; KONAT; VISSER, 2014b), Onion (ERDWEG et al., 2013; ERDWEG et al., 2015), and Whole (ERDWEG et al., 2013; ERDWEG et al., 2015) also provide extensive coverage of DSL development features.

Six LWs support both graphical and textual notations. Among them, GEMOC Studio and MPS are two non-commercial LWs that are actively maintained. GEMOC Studio stands out as the only tool with bidirectional model representation, allowing real-time synchronization of changes between the textual and graphical models. GEMOC Studio is built on EMF, Xtext, and Sirius, making it one of the most comprehensive open-source tools. Additionally, GEMOC Studio supports semantic execution and model simulation. MPS, on the other hand, covers graphical, textual, and tabular specifications (feature 03), but the implementation of the graphical representation of models (feature 02) is only partially completed, as mentioned in (ERDWEG et al., 2015).

Argyle is marked as covering graphical notation due to its support for graphical representations of Software Product Line specifications as feature models.

Two other LWs that support both graphical and textual notations, Enso and Más, are no longer being maintained or supported. Lastly, MetaEdit+, despite its completeness, does not provide a specification for textual notation, but it offers a textual representation of graphic models.

4.3 Conclusion

Our results indicate that there are only a few tools supporting a bidirectional DSL transformation between different workbenches. Besides, we also concluded that few DSL tools support bi-directional/multiple notations. Bi-directional DSLs tools embrace in the same tool more than one notation, including graphical, textual, symbolic, and/or tabular DSL notation. Likewise, another less explored tool feature is the support for customized graphical elements to represent concepts of the language application domain (*e.g.* `.SVG, .EPS, .JPG`), which helps improve language expressiveness in relation to the domain.

However, we were able to identify the necessary tools to carry out our work, dividing it into development stages where each stage can be considered an independent module but functions as a whole. The first module, the metamodel, was developed using Ecore tools; the second module, the graphical interface, was developed using Sirius Workbench, and the third module, code generation, was developed using Acceleo.

A comprehensive overview of this work, including details of the entire design and execution process, as well as full data analysis, can be found in the publication by (IUNG et al., 2020).

5 GRAPHICAL DOMAIN-SPECIFIC MODELING LANGUAGE FOR PERFORMANCE TESTING IN WEB SYSTEMS

In this chapter, the implementation of the OpenMLPerf Domain-Specific Language (DSL) will be presented. Various aspects will be discussed, ranging from the definition of requirements and design decisions to the implementation of the language using the Sirius LW. Section 5.1 covers the requirements modeled according to the performance testing domain and design decisions. Section 5.2 presents the metamodel and the underlying architecture of the DSL. Section 5.3 showcases the language implementation using the Sirius framework. The DSL code generation is discussed in the Section 5.4. Finally, Section 7.1 discusses the lessons learned from the development of the DSL.

5.1 Requirements and Design Decisions

The domain analysis, along with its formalization through an ontology, was defined in the work of (BERNARDINO, 2016). The ontology provides a foundation for determining concepts and relationships that represent the domain. Although the requirements baseline was also created in this work, improvements and changes have been made. The design decisions were based on the needs imposed by these requirements. In this section, the new requirements and design decisions for the DSL will be presented.

5.1.1 Language Requirements

This section lists the new requirements included in the OMLPerf proposal. The requirements baseline was previously established in the Requirements Analysis and Design Decisions for OpenMLPerf study (BERNARDINO, 2016).

Requirement (RQ)1) *The language should be implemented and made available under an open-source license.*

As a key change from the first version of OpenMLPerf, any tool used in the implementation of the language should be open-source, allowing it to be made available in repositories open to the community, academia, and industry.

RQ2) *The DSL should provide a graphical representation of performance testing features.*

This requirement pertains not to the language itself but to the tool used for its creation. Therefore, the tool should be capable of providing mechanisms to graphically represent all the features of the domain.

RQ3) *The DSL should generate code for testing tools, such as JMeter.*

In addition to providing a textual representation to facilitate the adoption and documentation of the language, the DSL should also be capable of automatically generating the necessary code for executing performance tests in specific tools, such as JMeter.

This will enable test engineers to use the language to define test scenarios more efficiently and effectively, eliminating the need to manually write code for the testing tools.

RQ4) *The DSL should enable traceability between elements in graphical and textual notations.*

To ensure clarity and consistency in the modeling process, it is essential that the DSL supports traceability between the graphical and textual representations of the performance test elements. This means that changes made in one representation should be automatically reflected in the other, allowing for seamless synchronization and maintaining the integrity of the test models. This traceability feature enhances the usability of the language and improves the overall efficiency of test development and maintenance.

5.1.2 Design Decisions

This section describes the design decisions made for the creation of the new version of the language, related to the requirements mentioned in Section 5.1.1. For each decision, its respective requirement(s) will be presented.

Design Decision (DD)1 *The use of open-source solutions for aiding the implementation of graphical DSLs* (RQ1, RQ2).

These requirements were fulfilled through a literature review conducted, presented in Chapter 4, where several tools that met these requirements were found. Therefore, it was decided to use the EMF and Eclipse Sirius frameworks, which are open-source and released under the Eclipse Public License 2.0 and Eclipse Public License 1.0, respectively. These frameworks provide support for creating DSLs with graphical notations.

DD2) *Enable support and traceability between graphical and textual notations* (RQ2, RQ3, RQ4).

This requirement was achieved by selecting the Eclipse development tools EMF and Sirius, which allow for the integration of graphical notation with the implementation of textual notation for the DSL. Additionally, the Acceleo tool was used for code generation from the models created with the DSL. The LW allows for the conversion and translation of model rules, providing bidirectional modeling between graphical elements and their corresponding assets in the textual notation. This means that a graphical notation can be mapped to multiple textual instances, ensuring traceability and consistency between the different representations of the DSL.

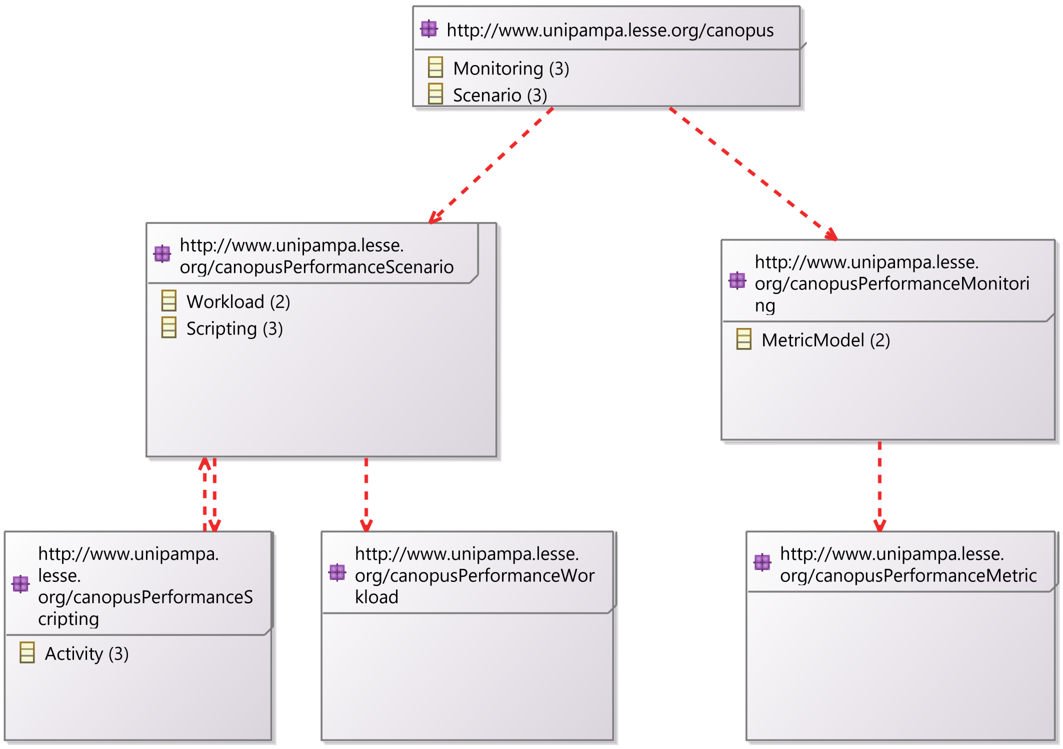
5.2 Architecture and Domain Modeling with Eclipse Ecore

In this section, the conceptual modeling or metamodel of the performance testing domain will be presented. The modeling was done using the Eclipse Modeling Framework (EMF) framework and its Ecore metamodel, through the Ecore Tools tool, a plugin installed directly from the Eclipse Marketplace.

The metamodel of the language was designed based on the requirements and design decisions presented in Section 5.1. The language is composed of two parts: *Scenario*, *Monitoring*, and *Scripting*. These concepts were abstracted from the performance testing domain.

Figure 8 shows a dependency diagram that illustrates how the language composition was defined, with two main metamodels: the *OpenMLPerf Performance Monitoring* and *OpenMLPerf Performance Scenario*. Together, they form the main model of the language, called Canopus. In addition to these two metamodels, there are three secondary metamodels that provide support to the main metamodels: *OpenMLPerf Performance Metric*, *OpenMLPerf Performance Workload*, and *OpenMLPerf Performance Scripting*.

Figure 8 – Package dependency diagram showing the main model and the six language metamodels


http://www.unipampa.lesse.org/canopus
Monitoring (3)
Scenario (3)
http://www.unipampa.lesse.org/canopusPerformanceScenario
Workload (2)
Scripting (3)
http://www.unipampa.lesse.org/canopusPerformanceMonitoring
MetricModel (2)
http://www.unipampa.lesse.org/canopusPerformanceScripting
Activity (3)
http://www.unipampa.lesse.org/canopusPerformanceWorkload
http://www.unipampa.lesse.org/canopusPerformanceMetric

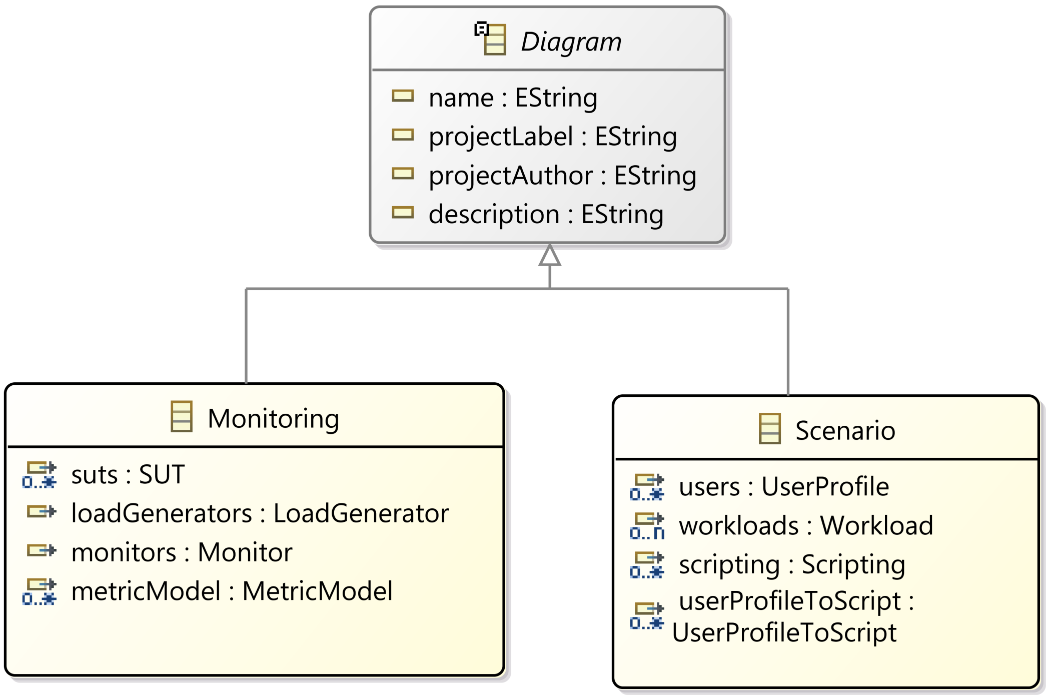
Source: Author

The *scenario* and *monitoring* objects are instantiated within a 'Canopus' object, representing the project itself. Thus, a project can encompass one or multiple scenarios and one or multiple monitoring instances. Additionally, various derived objects such as *script*, *workload*, and *metric* can be created. Figure 9 provides a visual representation of how the *workload* and *script* models are instantiated within the scenario, and the *metric* model is instantiated within the monitoring model.

Both the monitoring and scenario models serve as diagrams for defining the elements of a performance test. In each model, entities such as the System Under Test (SUTs), monitors, load generators, and metrics are instantiated to facilitate the test.

Furthermore, the workload, user profiles, and test scripts can be defined and instantiated within a specific test scenario

Figure 9 – Class diagram of the OpenMLPerf package



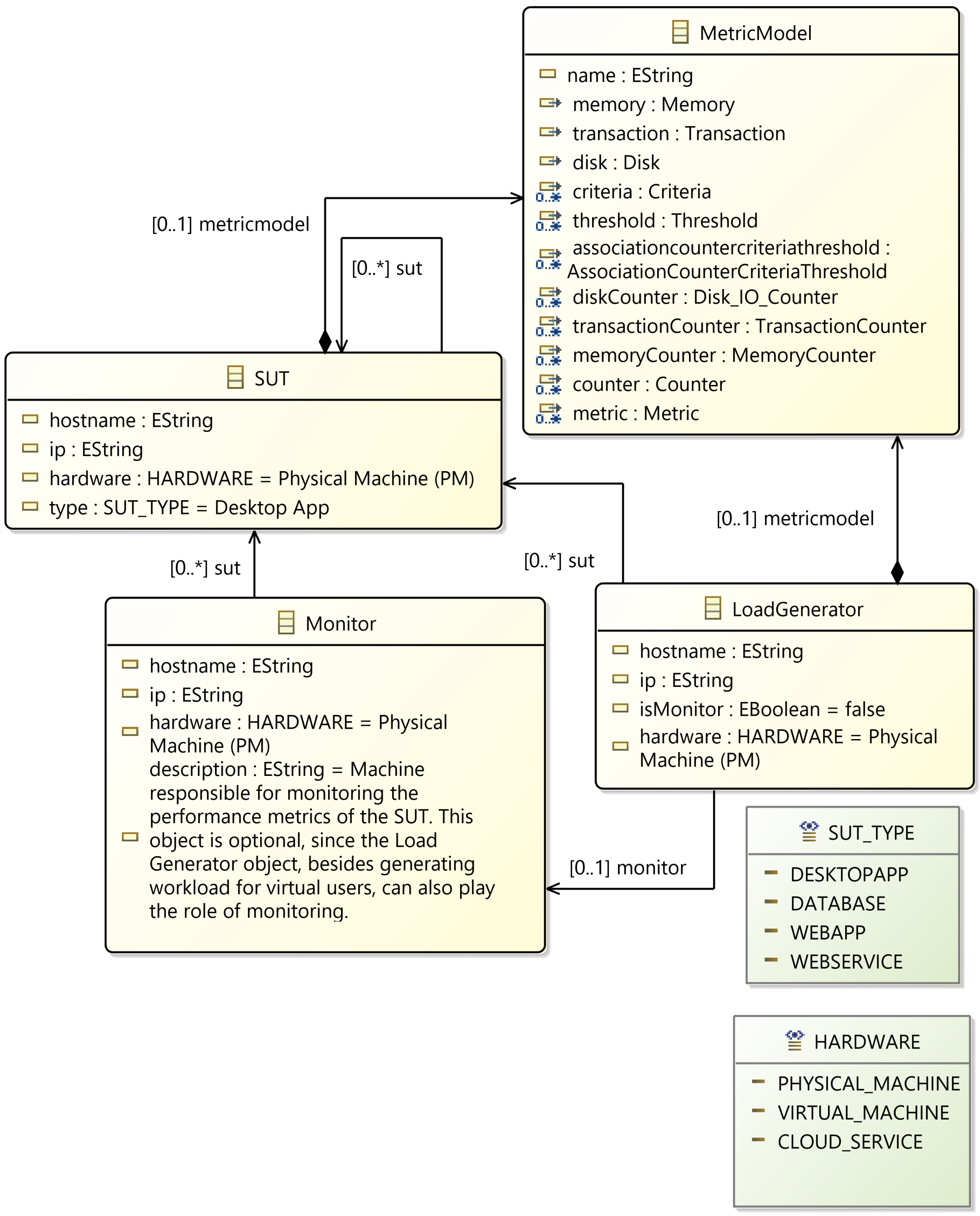
Source: Author

OpenMLPerf Performance Monitoring Metamodel

The *OpenMLPerf Performance Monitoring* metamodel (see Figure 10) is responsible for defining the elements related to the performance testing environment. It represents the System Under Test (SUT), which can be desktop or web applications, load generators, and monitoring machines, which can be physical, virtual, or cloud-hosted. Two enumerators are used to define the type of SUT and hardware, providing an efficient approach for selecting options from a list, whether it be a single choice or multiple alternatives. Figure 10 illustrates the possible association between multiple SUTs, as well as the association of a monitor with multiple SUTs for monitoring purposes. A load generator can also be associated with multiple SUTs and can be linked to a monitor as well.

This metamodel establishes the requirement of at least one SUT, one load generator, and one monitor to complete the modeling. The load generator can also be responsible for monitoring the scenario. The SUT should have a set of associated metrics that can be decomposed into another metric model. The metric model is defined by the *OpenMLPerf Performance Metric* metamodel.

Figure 10 – Class diagram illustrating the Performance Monitoring metamodel of the language


MetricModel
name : EString
memory : Memory
transaction : Transaction
disk : Disk
criteria : Criteria
threshold : Threshold
associationcountercriteriathreshold : AssociationCounterCriteriaThreshold
diskCounter : Disk_IO_Counter
transactionCounter : TransactionCounter
memoryCounter : MemoryCounter
counter : Counter
metric : Metric
[0..1] metricmodel
[0..*] sut
SUT
hostname : EString
ip : EString
hardware : HARDWARE = Physical Machine (PM)
type : SUT_TYPE = Desktop App
[0..*] sut
[0..*] sut
[0..1] metricmodel
Monitor
hostname : EString
ip : EString
hardware : HARDWARE = Physical Machine (PM)
description : EString = 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.
LoadGenerator
hostname : EString
ip : EString
isMonitor : EBoolean = false
hardware : HARDWARE = Physical Machine (PM)
[0..1] monitor
SUT_TYPE
DESKTOPAPP
DATABASE
WEBAPP
WEBSERVICE
HARDWARE
PHYSICAL_MACHINE
VIRTUAL_MACHINE
CLOUD_SERVICE

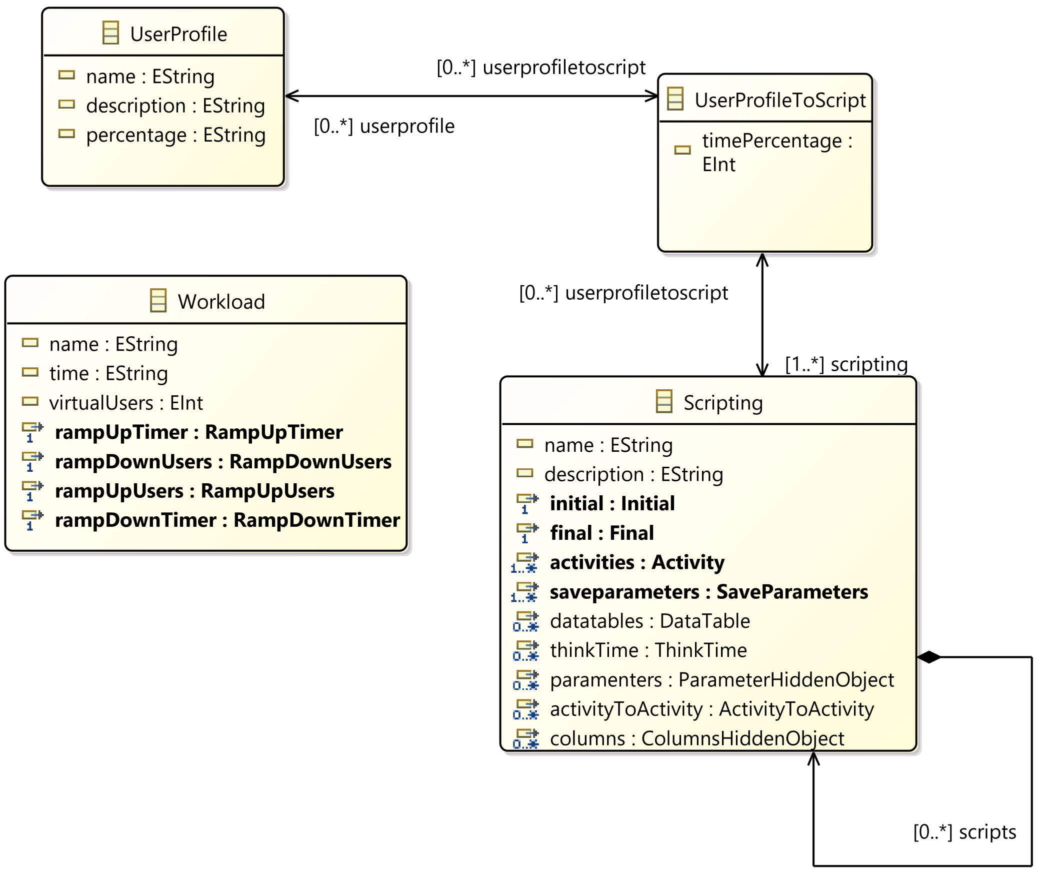
Source: Author

The *OpenMLPerf Performance Metric* (see Appendix A) establishes all the metrics to be monitored, their relationships, and counters. In this model, the values to be monitored in each of the metrics associated with the SUT are defined.

OpenMLPerf Perfomance Scenario Metamodel

The *OpenMLPerf Performance Scenario* metamodel (see Figure 11) is responsible for defining user profiles and their workload. It defines the flow of scripts that represent user activities, such as determining the percentage of time a user spends searching or purchasing on a website. It is necessary to define one of the workload profiles to determine which workload will be applied in the scenario. This metamodel has a relationship with the *OpenMLPerf Performance Scripting Metamodel*, as the scripts representing user activities are modeled in this metamodel.

Figure 11 – Class diagram depicting the metamodel of the performance scenario in the language



Source: Author

The *OpenMLPerf Performance Workload* metamodel (see Appendix A) defines the structure of workloads. It models the number of users in the system, the ramp-up (increase) or ramp-down (decrease) of this population, and the time interval.

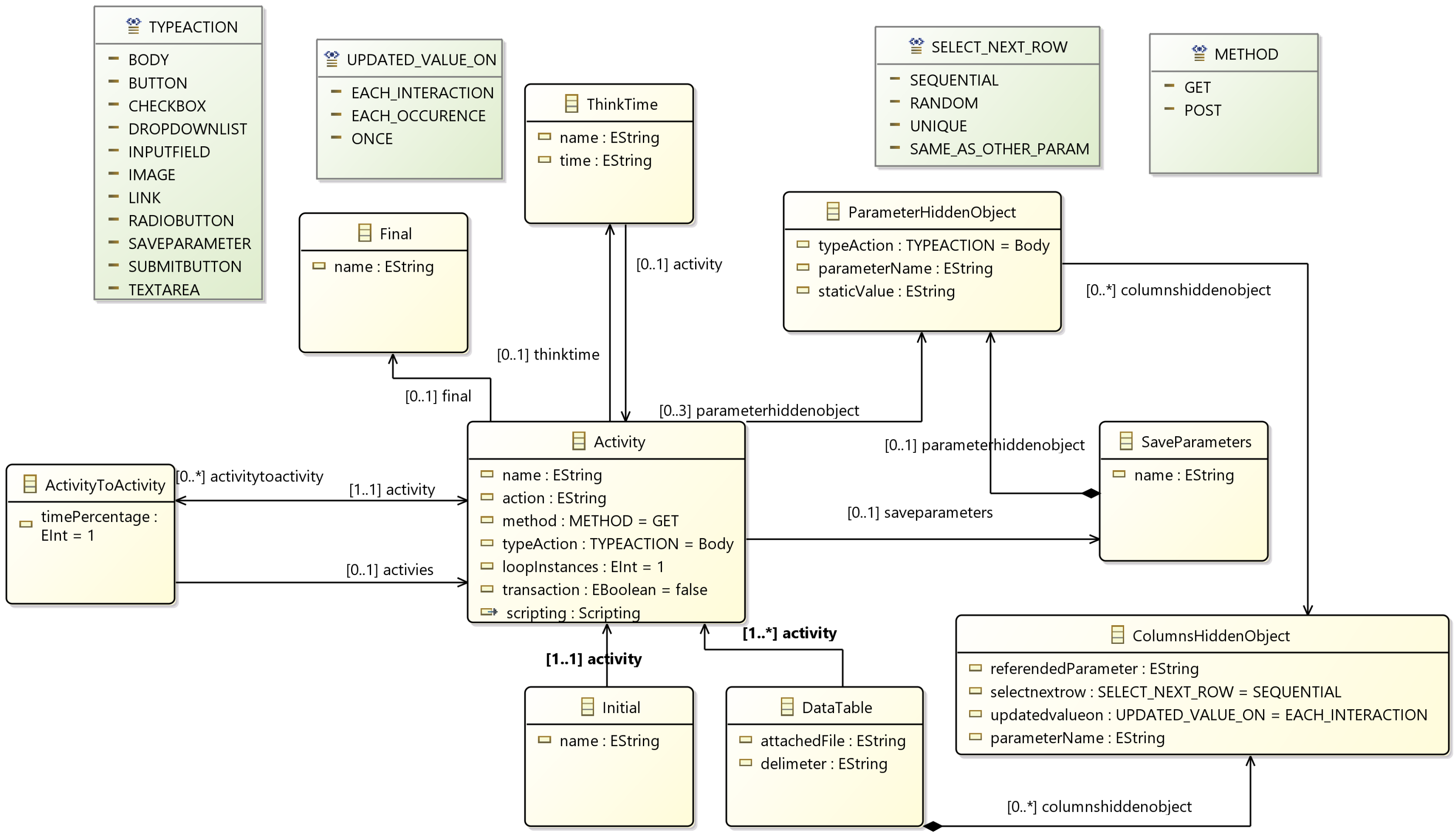
OpenMLPerf Performance Scripting Metamodel

The modeling of tasks and activities performed by the user profile is captured in the OpenMLPerf Performance Scripting metamodel (see Figure 12). This metamodel defines the interactions between the user profile and the Systems Under Test (SUTs). Each script consists of an initial and final milestone. In addition to activities, the metamodel includes elements such as Think Time, which represents the time between the task being available to the user and the execution time of that task, and Data Table, used for retrieval and storage of information for parameterization. Parameters can be saved for reuse in other script models. The metamodel also employs the strategy of using enumerators to define lists of characteristics for various model properties. The supporting metamodel for OpenMLPerf Performance Scripting, the OpenMLPerf Performance External File, is used to define the persistence and retrieval of relevant information for the scripts.

5.3 Language Implementation with the Eclipse Sirius Framework

This section presents the implementation of the language using the *Eclipse Sirius* framework. After creating a metamodel with the *EMF* framework and its *Ecore* metamodel, specifications need to be created for the elements and properties of the metamodel. These specifications define the graphical representations of the language. Specifications are created for everything that needs to be represented, including diagrams, objects (e.g., monitors, SUTs, scripts) as nodes, and relationships (e.g., connections between SUTs and monitors, relationships between activities in a script).

Figure 12 – Class diagram displaying the scripting metamodel of the language

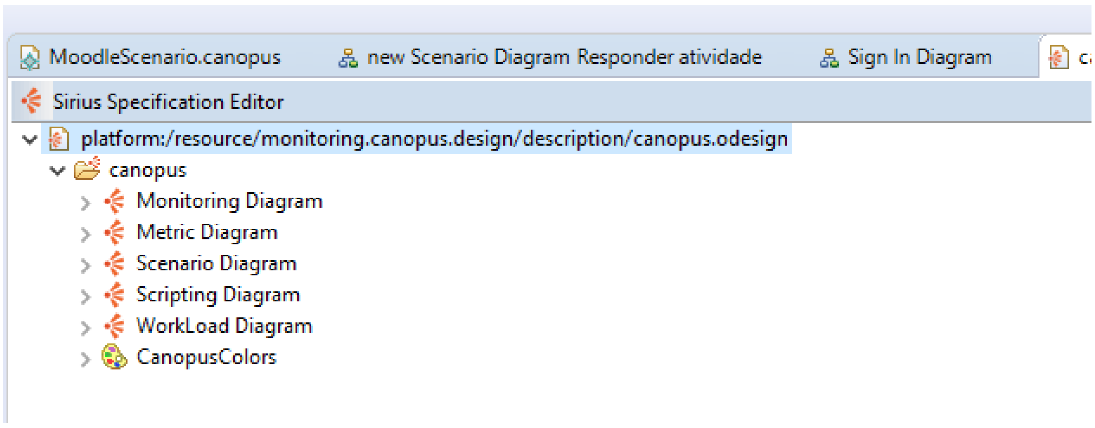


Source: Author

Node Definitions

For each model based on the metamodel that requires a graphical representation, a viewpoint specification or specification model is created in the Sirius editor. As shown in Figure 13, three viewpoint specifications have been created, one for each of the main models defined in the language metamodel. This allows each of these models to be represented, edited, and modeled within the modeling framework, providing an interactive experience for the end user.

Figure 13 – Sirius Specification Editor and the viewpoint specifications related to the models


MoodleScenario.canopus | new Scenario Diagram Responder atividade | Sign In Diagram | c

Sirius Specification Editor

platform:/resource/monitoring.canopus.design/description/canopus.odesign
- canopus
  - Monitoring Diagram
  - Metric Diagram
  - Scenario Diagram
  - Scripting Diagram
  - WorkLoad Diagram
  - CanopusColors

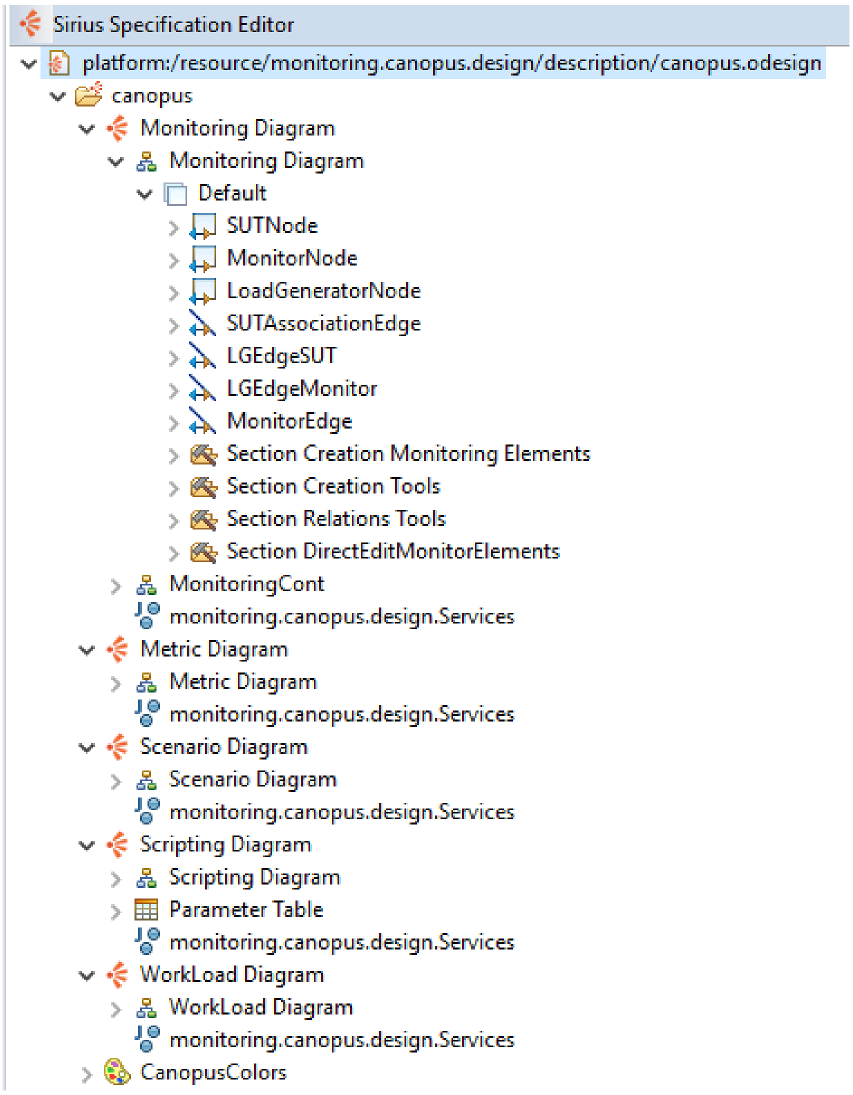
Source: Author

In each viewpoint specification, it is possible to create multiple representations of the same model, called descriptions, which can be diagrams, editing tables, cross tables, trees, and sequence diagrams. As shown in Figure 14, a diagram description was created to represent the monitoring model (*Monitoring Diagram*).

The same principle is used to create a representation for each element of the model that requires a graphical representation. For this purpose, a node is created and assigned to a semantic element of the model. In Figure 14, we can see the three main elements of the monitoring model: SUTNode, representing the SUT in the monitoring diagrams; MonitoringNode, representing the monitoring servers; and LoadGenerator, responsible for representing the load generator servers. When defining the nodes, it is possible to assign images and icons instead of the default EMF icons. Some of the elements in the work have icons in the representation, but they are only used for testing purposes and are not final.

In Figure 15, the viewpoint specification for the *OpenMLPerf Specification Metric* model is presented. This metrics model is associated with a SUT. When associated, the model is expanded into a diagram for modeling these metrics, which can also be associated with multiple SUT or monitoring models.

Figure 14 – Modeled nodes to represent the elements of the *OpenMLPerf Specification Monitoring* metamodel



```
Sirius Specification Editor
platform:/resource/monitoring.canopus.design/description/canopus.odesign
  canopus
    Monitoring Diagram
      Monitoring Diagram
        Default
          SUTNode
          MonitorNode
          LoadGeneratorNode
          SUTAssociationEdge
          LGEdgeSUT
          LGEdgeMonitor
          MonitorEdge
          Section Creation Monitoring Elements
          Section Creation Tools
          Section Relations Tools
          Section DirectEditMonitorElements
      MonitoringCont
      monitoring.canopus.design.Services
    Metric Diagram
      Metric Diagram
      monitoring.canopus.design.Services
    Scenario Diagram
      Scenario Diagram
      monitoring.canopus.design.Services
    Scripting Diagram
      Scripting Diagram
      Parameter Table
      monitoring.canopus.design.Services
    WorkLoad Diagram
      WorkLoad Diagram
      monitoring.canopus.design.Services
    CanopusColors
```

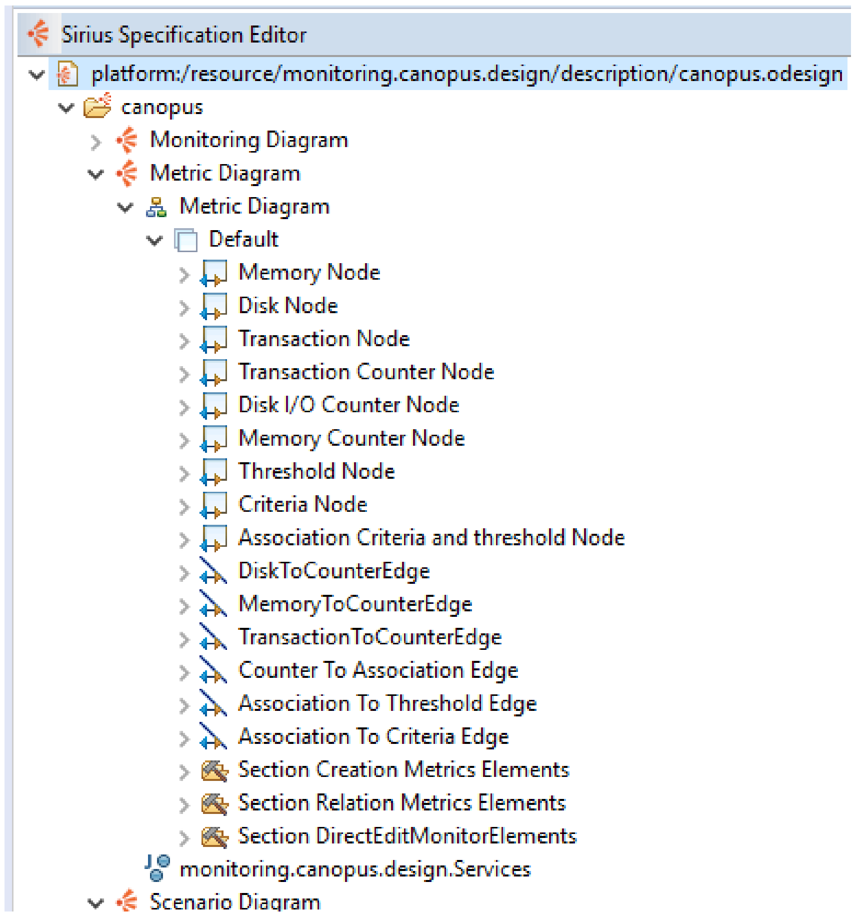
Source: Author

5.3.1 Definition of Relationships

To define relationships, an element of type *Edge* is created and assigned to a semantic element that represents the relationship between classes in the metamodel. The source and target of the relationship are defined, indicating the starting and ending points of the relationship. Arrow types, source endpoint elements, and target endpoint elements can also be defined. In this work, relationship elements were defined to express the direction and differentiation of each relationship.

In Figure 16, you can see the differences between the elements of relationships between the SUTs, as well as between the load generator and the SUTs.

Figure 15 – Modeled nodes to represent the elements of the *OpenMLPerf Specification Metric* metamodel



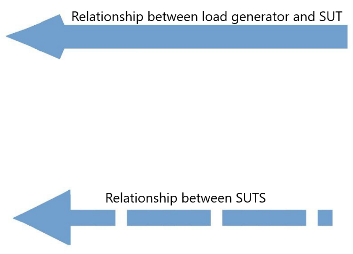
```
Sirius Specification Editor
platform:/resource/monitoring.canopus.design/description/canopus.odesign
  canopus
    Monitoring Diagram
    Metric Diagram
      Metric Diagram
        Default
          Memory Node
          Disk Node
          Transaction Node
          Transaction Counter Node
          Disk I/O Counter Node
          Memory Counter Node
          Threshold Node
          Criteria Node
          Association Criteria and threshold Node
          DiskToCounterEdge
          MemoryToCounterEdge
          TransactionToCounterEdge
          Counter To Association Edge
          Association To Threshold Edge
          Association To Criteria Edge
          Section Creation Metrics Elements
          Section Relation Metrics Elements
          Section DirectEditMonitorElements
      monitoring.canopus.design.Services
    Scenario Diagram
```

Source: Author

5.3.2 Modeling Diagrams

The Figure 17 presents the monitoring diagram, where it is possible to specify all the information related to performance testing monitoring, as well as the metric associated with the element (green square box next to the element) of both the SUT and the load generator. It is also possible to define identifiers such as IP addresses, names, and

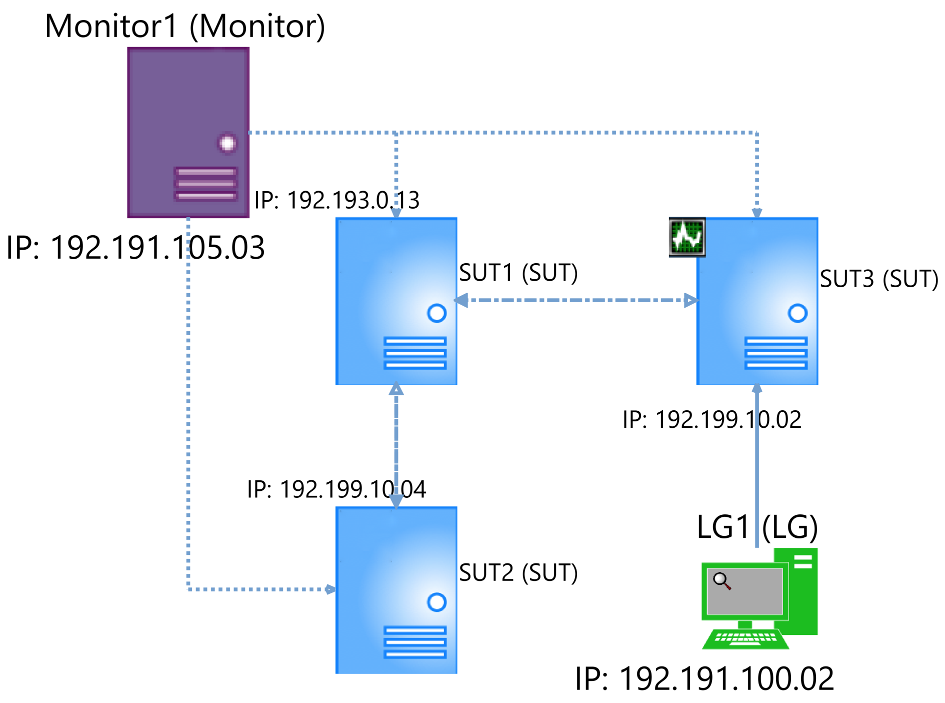
Figure 16 – Example of graphical relationship elements (edges) between objects in the model.


Relationship between load generator and SUT
Relationship between SUTS

Source: Author

information about the type of machine and system, whether physical, cloud, or virtual.

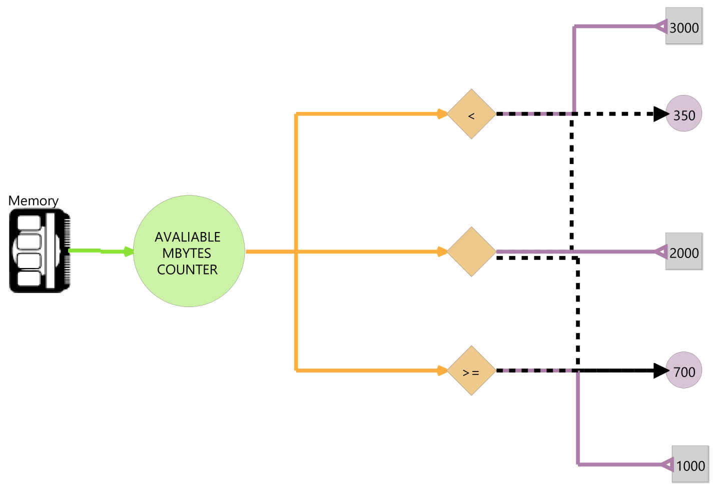
Figure 17 – Monitoring Test Modeling Diagram



Source: Author

In Figure 18, wis presented the Metrics Diagram of the monitoring, where the metrics to be monitored during the test are defined. In this diagram, it is possible to specify the metric, define one or more counters to be monitored, determine the values and intervals to be monitored for each metric, and create criteria for the metric values.

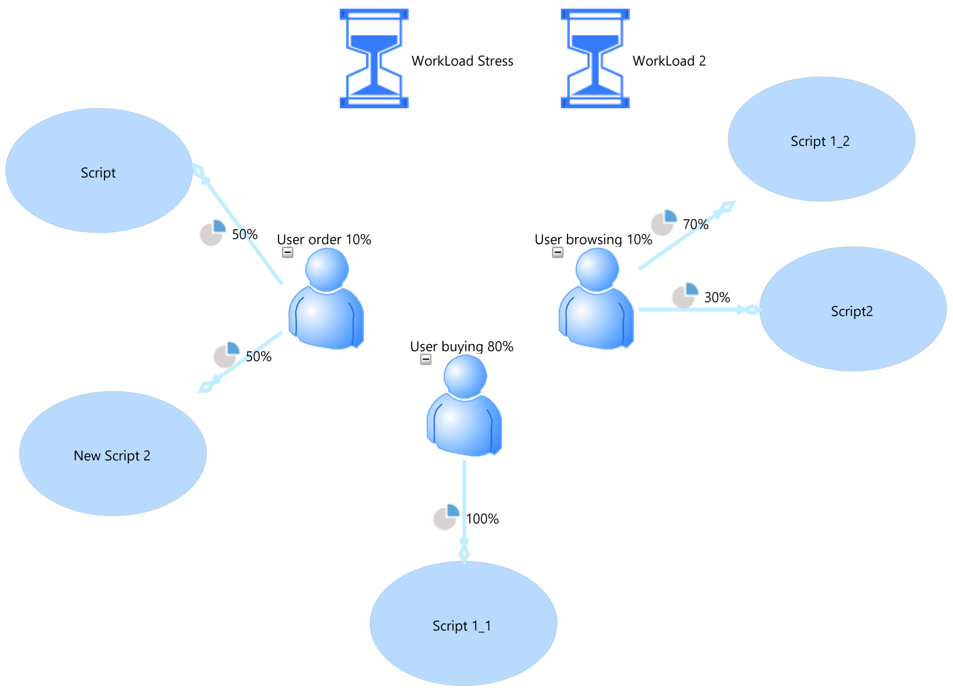
Figure 18 – Monitoring Metric Modeling Diagram



Source: Author

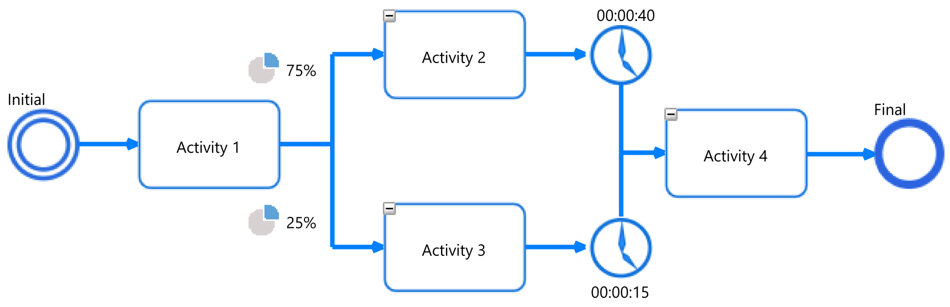
As shown in Figure 19, the scenario modeling diagram allows us to visually specify a test scenario, including user profiles, scripts, and workloads. It enables the definition of the percentage of time allocated to each profile in a particular activity and allows the reuse of profiles and scripts in multiple scenario models.

Figure 19 – Test Scenario Modeling Diagram



Source: Author

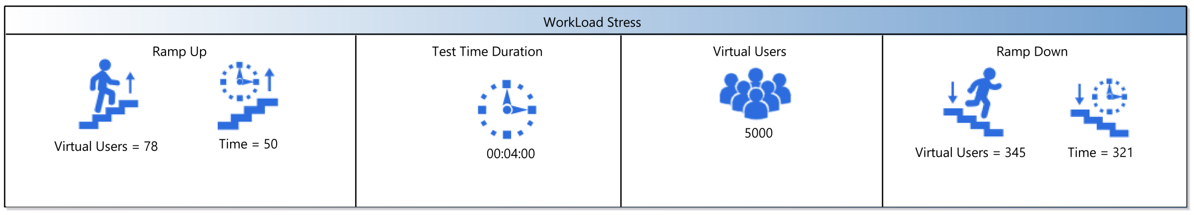
Figure 20 – Modeling diagram of a scenario script



Source: Author

The diagram for script specification, as shown in Figure 20, allows you to define all the activities that simulate user interaction with the system. You can specify the time it takes to start each activity, the probability of the user performing one activity over another, and create parameters for each activity. For example, if the activity involves clicking a button, filling out a field, or loading a form. You can also store parameters or groups of parameters for reuse in other activities and scripts. The workload diagram allows you to define the configuration of performance characteristics in a test scenario. As shown in Figure 21, you can specify the test duration, the number of virtual users in the test, and the user arrival and departure rate within a specific time interval.

Figure 21 – Workload modeling diagram



Source: Author

5.3.3 Modeling Menus

The Sirius framework also allows the specification of menus to be used by users in diagram modeling. The side menu functions as a customized palette, where users can view all the elements that can be created in the diagram, as well as all the allowed relationships. In Figure 22 and Appendix B, you can observe how the menus for the DSL are defined. Menu specification also includes the addition of contextual menus, which appear alongside the mouse pointer on the modeling diagram. In contextual menus, users can add elements to all test diagrams, just like in the side menus. The only representation that cannot be established through contextual menus is the relationship representation between elements. However, apart from using the side menus, it can also be established through property editing menus.

The property editing menus, as shown in Figure 23, allow users to edit attributes related to the elements of the diagram and associated models. Users can modify components, edit appearance, and establish new relationships between various models of the same metamodel, as long as these relationships are already specified in the metamodel.

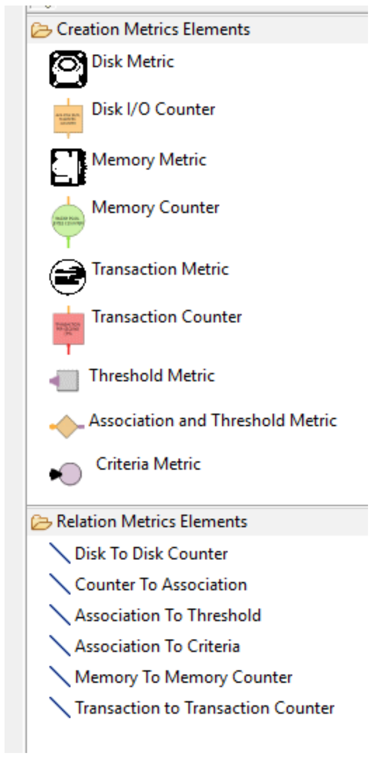
5.4 Code Generation

The code generation phase of the OpenMLPerf DSL involves the transformation of the performance testing models created using the DSL into executable code artifacts. This process is facilitated by integrating the Acceleo code generator, which provides powerful capabilities for template-based code generation.

5.4.0.1 Acceleo Template Development

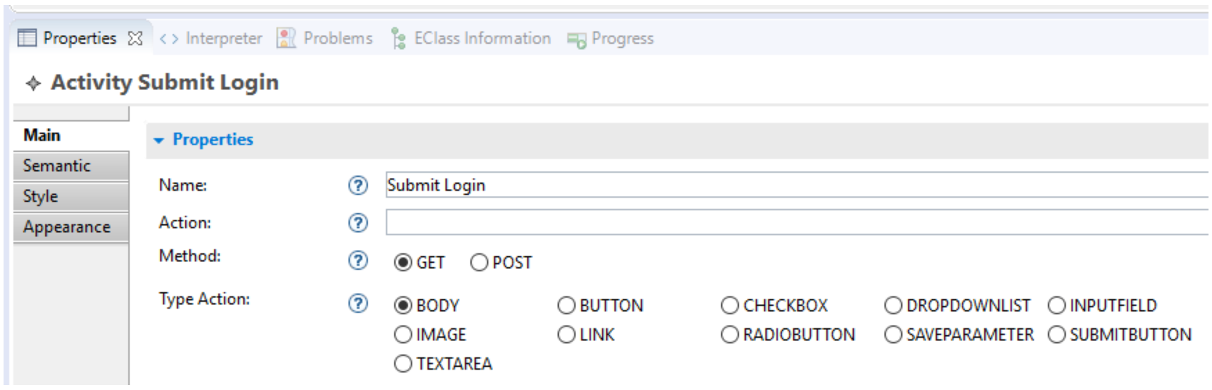
To generate code from the OpenMLPerf models, we leverage the Acceleo framework to define templates that specify how the models should be transformed into code. These

Figure 22 – Menu for creating the Scripting diagram



Source: Author

Figure 23 – Properties editing menu



Activity Submit Login

Name: Submit Login

Action:

Method: [✓ GET] POST

Type Action: [✓ BODY] BUTTON CHECKBOX DROPDOWNLIST INPUTFIELD IMAGE LINK RADIOBUTTON SAVEPARAMETER SUBMITBUTTON TEXTAREA

Source: Author

templates consist of a combination of static text and dynamic expressions written in the Acceleo language.

The first step in implementing the code generation process is to develop the Acceleo templates. These templates, which is presented in the Figure 5.1 define the structure and content of the generated code artifacts, including the necessary components, functions, and configurations for executing performance tests. The templates can be tailored to the specific requirements of the target system and the performance testing scenarios being modeled.

In the context of OpenMLPerf, the Acceleo templates should be designed to handle the various elements of the DSL, such as performance testing scenarios, metrics, workloads, and other related components. The templates should extract relevant information from these models and use it to generate the corresponding code representations.

5.4.0.2 Integration with OpenMLPerf DSL

The integration of the Acceleo code generator with the OpenMLPerf DSL involves establishing a seamless connection between the DSL models and the Acceleo templates. This integration allows for the automatic transformation of DSL models into executable code.

In the implementation, the DSL models serve as the input for the Acceleo code generator. The generator processes these models and applies the defined templates to produce the desired code output. The generator utilizes the information contained in the DSL models to determine the appropriate code generation logic, ensuring that the generated code accurately reflects the specified performance testing scenarios.

To facilitate this integration, it is necessary to establish mappings between the elements of the DSL models and the corresponding sections of the Acceleo templates. This mapping ensures that the relevant information is extracted and correctly used in the code generation process. Additionally, any necessary transformations or computations required for code generation can be implemented within the Acceleo templates.

5.4.0.3 Customization and Extension

One of the advantages of using Acceleo for code generation is the flexibility it provides for customization and extension. The generated code artifacts can be tailored to meet specific requirements and constraints of the target system and performance testing scenarios.

In the context of OpenMLPerf, customization may involve configuring the generated code to accommodate specific hardware or software environments, setting up performance monitoring libraries or tools, and defining integration points with other system components. These customizations can be implemented within the Acceleo templates, allowing for a high degree of flexibility in adapting the generated code to different testing contexts. Furthermore, the Acceleo code generator allows for the extension of the templates to incorporate additional functionality or support new features. This extensibility enables the OpenMLPerf DSL to evolve and adapt as new requirements arise in the field of performance testing.

5.4.0.4 Generated Code Artifacts

The output of the code generation process is a set of executable code artifacts that can be used to conduct performance tests. These artifacts encompass the necessary scripts, configurations, and components to simulate user interactions, generate workloads, collect metrics, and analyze system performance.

The generated code artifacts should adhere to best practices and coding standards to ensure their quality, maintainability, and reusability. Proper documentation should be provided to guide users in executing the generated code and interpreting the results of the performance tests. It is important to note that while the Acceleo code generator automates the process of generating code from the OpenMLPerf DSL models, it is still necessary to validate and verify the generated code.

5.4.0.5 Generation of Scala Code for JMeter

One specific aspect of the code generation process with Acceleo in the context of OpenMLPerf is the generation of Scala code for the JMeter tool. This feature allows for the automatic transformation of performance testing models into Scala code that can be executed by JMeter, a popular open-source load testing tool. By leveraging the Acceleo templates, the OpenMLPerf DSL can generate Scala code that defines JMeter test plans, including HTTP requests, assertions, and other necessary configurations. This integration provides users with the ability to seamlessly generate JMeter scripts directly from the DSL models, reducing manual effort and ensuring consistency between the models and the generated code. The Acceleo code generator presented in the Figure 5.1, is invoked with the following parameters:

- `generate` module: Specifies the metamodels to be used in the code generation process. The OpenMLPerf metamodels are imported and used to define the structure and behavior of the generated code. The metamodels include:
  - `openmlperf`
  - `openmlperfPerformanceExternalFile`
  - `openmlperfPerformanceMetric`
  - `openmlperfPerformanceMetricMonitoring`
  - `openmlperfPerformanceScenario`
  - `openmlperfPerformanceWorkload`
  - `openmlperfPerformanceScripting`
- `org::unipampa::openmlperf::acceleo::services::services` import: Imports the necessary Acceleo services for performing code generation tasks.

The main entry point for code generation is the `generateElement` template, which takes a scenario as a parameter and generates the corresponding Scala code. The generated code is then written to a Scala source file with the same name as the scenario, but with the `.scala` extension.

The generated Scala code follows the Gatling DSL (Domain-Specific Language) for load testing. It defines a class that extends the `Simulation` class provided by Gatling. The class name is derived from the scenario name, with the first letter capitalized. Within the generated class, individual Gatling scenarios are defined for each scripting element in the scenario model. These scenarios correspond to the activities performed during the load testing, such as sending HTTP requests and processing responses. For each scripting element, a scenario object is created with a name matching the scripting element's name. Within the scenario object, HTTP requests are defined using the `exec` method, which specifies the request method, action, and any associated parameters. The details of the requests are derived from the activities defined within the scripting element.

Additionally, the generated code includes sections for defining user profiles and workloads. These sections are currently commented out, as they require further customization based on your specific requirements. The generated code also includes the necessary configuration for the Gatling HTTP protocol, such as the base URL, headers, and other settings. These configurations can be customized as needed for your load testing scenarios.

The generated Scala code for JMeter can be executed within the JMeter environment, enabling users to conduct performance tests based on the defined scenarios and workloads specified in the OpenMLPerf DSL. This integration enhances the usability and practicality of the OpenMLPerf framework by leveraging the capabilities of JMeter and Scala for performance testing. It is important to note that while the Acceleo code generator automates the process of generating code from the OpenMLPerf DSL models, users should still validate and verify the generated code to ensure its correctness and suitability for their specific testing requirements. Proper understanding of JMeter and Scala is necessary to make the most effective use of the generated code artifacts.

```
[comment encoding = UTF-8 /]
  [module generate('http://www.unipampa.lesse.org/openmlperf',
     'http://www.unipampa.lesse.org/
     openmlperfPerfoamnceExternalFile', 'http://www.unipampa.
     lesse.org/openmlperfPerformanceMetric', 'http://www.
     unipampa.lesse.org/openmlperfPerformanceMetricMonitoring',
     'http://www.unipampa.lesse.org/
     openmlperfPerformanceScenario', 'http://www.unipampa.lesse.
     org/openmlperfPerformanceWorkload', 'http://www.unipampa.
     lesse.org/openmlperfPerformanceScripting')]
  [import org::unipampa::openmlperf::acceleo::services::
   services/]

[template public generateElement(aScenario : Scenario)]
[comment @main/]
[file (aScenario.name.concat('.scala'), false, 'UTF-8')]
package [aScenario.projectLabel.toLower().trim()/]

import scala.concurrent.duration._
import io.gatling.core.Predef._
import io.gatling.http.Predef._
import io.gatling.jdbc.Predef._

class [aScenario.name.toUpperFirst().trim()/] extends
   Simulation{
[for (scripting : Scripting | aScenario.scripting)]
    object [scripting.name/] {
[for (datatable : DataTable | scripting.datatables)]
[retornaTable(datatable)/]
[/for]
    val [scripting.name.toLowerCase()/] = scenario("
       RecordedSimulation")
                    [for (activity : Activity| scripting.
                       activities)]
                    .exec(http("request_0")
                    .[activity.method.toString().toLowerCase
                       ()/]("[activity.action/]")
                    [returnTableParam(activity)/]
                    [/for]
    }
[/for]
[for (user : UserProfile | aScenario.users)]
    //[user.name/] [user.percentage/]%
[/for]
[for (work : Workload | aScenario.workloads)]
    //[work.name/] : [work.virtualUsers/] virtual users
            [work.rampUpTimer.time/]
            [work.rampUpUsers.virtualUsers/]
[/for]
val httpProtocol = http
            .baseUrl("[aScenario.baseURL/]")
            .inferHtmlResources()
            .acceptEncodingHeader("gzip, deflate")
```

```
                .acceptLanguageHeader("pt-BR,pt;q=0.9,en-US;q=
                   0.8,en;q=0.7,sv;q=0.6")
                .userAgentHeader("Mozilla/5.0 (Windows NT 10.0;
                   Win64; x64) AppleWebKit/537.36 (KHTML, like
                   Gecko) Chrome/84.0.4147.125 Safari/537.36")
    val headers_0 = Map(
                "Accept" -> "text/html,application/xhtml+xml,
                   application/xml;q=0.9,image/webp,image/apng
                   ,*/*;q=0.8,application/signed-exchange;v=b3;q=
                   0.9",
                "Sec-Fetch-Dest" -> "document",
                "Sec-Fetch-Mode" -> "navigate",
                "Sec-Fetch-Site" -> "none",
                "Sec-Fetch-User" -> "?1",
                "Upgrade-Insecure-Requests" -> "1")
    }
    [/file]
    [/template]
```

Listing 5.1 – SCALA Template Generator

5.4.1 Chapter Lessons

In this chapter, we presented the requirements and design decisions for OpenMLPerf. Each requirement was discussed, and design decisions were presented to address them. We also described the implementation of the language, including the architecture structure using package diagrams created with EMF. EMF was used to create the language's metamodel, which defines the structural, conceptual, and relationship aspects that the language must adhere to. Additionally, we implemented a code generator for the DSL. The code generator translates the models created with OpenMLPerf into executable performance testing artifacts. This allows users to seamlessly move from modeling to execution, automating the generation of test scripts, workload configurations, and other necessary artifacts.

In the next chapter, we will present the evaluation conducted to assess the effort required for performance testing modeling using OpenMLPerf compared to a Unified Modeling Language (UML) profile for performance testing. We will discuss the results obtained from the modeling activities and a post-experiment survey, in which participants answered questions related to the effectiveness, ease of use, and intuitiveness of the approach. These insights will provide valuable feedback for further improvements and enhancements of OpenMLPerf.

6 EMPIRICAL EVALUATION

This chapter presents the results and discussions of an empirical evaluation conducted to assess the effort required to use OpenMLPerf. The evaluation compared the modeling efforts with OpenMLPerf against the modeling efforts using a UML-based approach for performance testing.

The empirical evaluation is organized as follows: Section 6.1 presents the experiment definition, research questions, and objectives. Section 6.2 describes the experiment planning, including the hypotheses, research questions, and variables. It also discusses the participant selection, evaluation design, and threats to the validity of the evaluation. In Section 6.3, the steps of preparation and execution during the experiment are discussed. Section 6.3.3 presents the findings of the evaluation. Finally, Section 6.3.4 discusses the results, execution, and improvements of the evaluation.

6.1 Evaluation Definition

This section presents the research questions, evaluation definition, and research objectives for the evaluation. It also outlines the hypotheses for the research questions, as well as the instruments and design of the evaluation.

6.1.1 Research Questions

The objective of this experiment is to gather quantitative or qualitative data regarding the effort required to model performance testing scenarios using OpenMLPerf and compare it with the effort required when using a UML profile (RODRIGUES et al., 2015) for performance testing. Creating test scenarios and the tests themselves can be one of the most time-consuming tasks during software development. Using activities related to MDE, such as MBT and DSLs, becomes a promising alternative in terms of time and cost savings. This approach provides the automatic generation of test scripts after modeling the scenarios, specific to the domain being tested, without the need to understand the technologies responsible for executing the tests. On the other hand, this approach requires a high level of knowledge about the domain being tested and the performance testing domain.

Thus, it was decided to conduct this experiment to gather information about the effectiveness, ease of use, and effort related to modeling performance tests using a DSL and using a UML profile. To do so, the research strategy chosen was to define the questions based on the tasks that would be assigned to the evaluation participants, simulating scenario modeling in a typical industry environment using OpenMLPerf and the UML profile. This way, the aim is to identify the effort required for both types of modeling and compare the two approaches.

The evaluation contains the following research questions (RQs) to be answered:

RQ1. *What is the effort required to model performance tests using a UML profile and OpenMLPerf?*

RQ2. *How effective is the modeling of performance tests using a UML profile and OpenMLPerf?*

RQ3. *How intuitive/easy is it to model performance tests using a UML profile and OpenMLPerf?*

6.1.2 Objective Definition

The research objectives of this evaluation are to measure the effort and intuitiveness in using OpenMLPerf and the UML profile for performance testing. The effort, measured in time, combined with the intuitiveness and ease, which is measured based on the users' opinions of the tools, during the modeling of test scenarios, help to understand the practicality of constructing models and the time required for it.

6.2 Planning

In this section, the evaluation planning, research questions, and their hypotheses will be presented. The selection of participants, experiment design, and threats to validity will also be discussed.

6.2.1 Context

The evaluation context can be divided into four concepts:

- **Process:** An in-vitro approach was used in a semi-controlled environment, as the participants used their personal machines without online activities.
- **Participants:** The participants are undergraduate students in computer science courses.
- **Reality:** The evaluation addresses real-world problems, such as modeling test scripts.
- **Generality:** This evaluation is conducted in a specific context, but the results can be applied to questions regarding the use of DSLs in different environments, i.e., environments configured for the use of DSLs.

6.2.2 Hypothesis Formulation

In this section, we will present the hypotheses and the measures used to evaluate the response to RQ1. The notations used for each hypothesis are as follows:

Φ_{dsl}: Represents the measure when using OpenMLPerf for modeling performance testing scenarios.

Φ_{uml}: Represents the measure when using the *UML* profile for modeling performance testing scenarios.

The research questions, previously presented, and their hypotheses are as follows:

+**RQ1.** *What is the effort required to model performance tests using a UML profile and OpenMLPerf?*

Null Hypothesis, H0: $\Phi dsl == \Phi_{uml}$: The effort is the same when using the *UML* profile and OpenMLPerf to create performance test models.

Alternative Hypothesis, H1: $\Phi dsl < \Phi_{uml}$: The effort is lower when using the *UML* profile compared to using OpenMLPerf to create performance test models.

Alternative Hypothesis, H2: $\Phi dsl > \Phi_{uml}$: The effort is higher when using the *UML* profile compared to using OpenMLPerf to create performance test models.

6.2.3 Participant Selection

The participants of the evaluation are undergraduate students in Software Engineering and Computer Science courses at the Federal University of Pampa (UNIPAMPA). Regularly enrolled students without any restrictions regarding the current semester in the course were invited to participate. To level the participants' knowledge, they had to respond to a survey (see Appendix D) with questions related to their level of knowledge in software modeling techniques, both with *UML* and *DSLs*. Questions were also asked about the participants' technical knowledge in performance testing and modeling languages. As a result of this survey, a balanced level of knowledge among the participants was identified, with no participants outside the knowledge curve of the group.

6.2.4 Evaluation Design

The evaluation framework adheres to the following guidelines:

Randomization: Participants were randomly assigned to perform the tasks using OpenMLPerf or the *UML* profile.

Grouping: Since there was no significant difference in the participants' knowledge levels, individuals were divided into two homogeneous groups.

Balancing: Participants were grouped into two groups, where each group was required to perform the task using one of the approaches.

6.2.5 Instrumentation

To support the participants in the evaluation, technical specifications, use cases, and performance requirements documents were provided. Additionally, two supporting tools were used, one for each approach. The Astah Professional modeling tool(ChangeVision,

2023) was used for modeling use cases and activity diagrams using the *UML* approach. An adapted version of the Eclipse IDE with support for the DSL metamodels was used for modeling the representation diagrams of the DSL models.

A training session was conducted, where the concepts related to OpenMLPerf and the *UML* profile for performance testing were presented to the participants. A practical tutorial demonstrating how to create performance models using the Astah tool and the *UML* profile was provided. Along with the tutorial, a manual with details about the *UML* profile and instructions on how to use the Astah modeling tool was provided. Similarly, a tutorial allowed participants to observe the application of OpenMLPerf for creating performance test models. A manual with information on how to install the necessary models for the execution of the language and how to perform performance test modeling was also provided.

In the experiment execution session, the participants interacted with a web application called Moodle. To perform the modeling tasks, the previously mentioned documents with detailed technical specifications of the tests were used. Based on these documents, the participants had to create the performance models.

The effort of each participant to complete the modeling task was collected, aiming to answer RQ1. To answer RQ1 and RQ2, data was collected through a survey conducted after the experiment.

6.2.6 Threats to Validity

In this section, the threats to the validity of the evaluation and the mitigating actions taken will be described. The classification scheme presented by Hyman (HYMAN, 1982), which categorizes threats into four types, was adopted in this evaluation. The following are the threats:

Conclusion validity: The participants' profile is a significant threat to the study, as there were no representatives from the industry, and the small group of academic participants did not have considerable experience with modeling languages, performance testing, and *UML* profiles. The researcher's bias when analyzing the results is also a threat to consider. To mitigate this threat, the effort was analyzed in a non-human way through statistical time data analysis. Another threat was conducting the evaluation on participants' personal machines instead of machines with the same configuration, such as lab machines. To mitigate this, a unique build of the DSL was developed, and the same version of Astah Professional was provided.

Internal validity: The training session and the evaluation session were conducted on separate days, which is a threat to the internal validity of the study. As there was a one-day gap between the two sessions, participants may forget what was presented in the training. To mitigate this, manuals and guides for each approach were distributed. To ensure balance between participant groups, a survey was conducted, allowing for sample grouping.

External validity: The selection of participants for the experiment is a threat to the evaluation, as it was not possible to invite participants with considerable experience to participate.

Construct validity: One threat to construct validity was using only one application and system for modeling, limiting the number of variables.

6.3 Evaluation Operation

In this section, the activities of preparation and execution of the evaluation operation will be discussed.

6.3.1 Preparation

The preparation of the experiment began with the selection of applications and possible scenarios to serve as a model for creating performance test scenarios. Specifications, requirements, and use case documents were made available to all participants, as well as manuals for both approaches and executable versions of the two modeling tools used, Astah and Eclipse IDE. Additionally, a survey was conducted to gather information and profile the participants. This allowed us to identify that there were no discrepancies in participants' knowledge levels, indicating a very homogeneous group.

6.3.2 Execution

The evaluation of the modeling feature and graphical notation was planned to be executed before the development of the code generator. This sequencing ensures that the code generator templates can be accurately tailored to the PT domain. A training session was held in the days leading up to the evaluation, during which both approaches were applied to offer an overview of both tools and address participants' questions. Each approach required a modeling task, specifically focused on the performance scenario, which needed to be expanded to include performance script modeling. During the training session, participants had to model a performance scenario representing the interaction between two user profiles and the LimeSurvey online survey system: the student profile and the professional profile, both responding to a survey. The scenario included a workload of one thousand virtual users, a test duration of four hours, and the execution of a script named "Answer the Survey", composed of 20 activities representing the user's interaction while answering each question of the survey.

Table 1 presents the distribution of participants in the execution of the approaches. Nine participants were divided into two groups, with five participants performing the tasks with the UML approach and four participants completing the tasks with the OpenMLPerf approach.

Approach	Participants
UML	5
OpenMLPerf	4

Table 1 – Number of participants per approach.

During the execution phase, participants performed a task using the assigned approach for each group. Each task can be described as follows:

- UML: Performance models were to be constructed according to the provided specifications using the UML profile for performance testing.
  - Scenario Modeling: In this task, each participant built a model based on the performance requirements using Astah. Use case diagrams could represent the performance scenario model and the user profiles, which included two actors: students and teachers. Three use cases, which could be represented with activity diagrams, could be associated with the scenario users.
  - Script Modeling: For the script task, participants could refer to the information found in the provided use case specifications. Performance-related information could be added to each activity within the script.
- OpenMLPerf: Performance models were to be constructed according to the provided specifications using OpenMLPerf for performance testing.
  - Scenario Modeling: To perform this task, each participant had to use OpenMLPerf and build OpenMLPerf Performance Scenario Models. The Eclipse IDE was used as the platform, with the addition of plugins containing the necessary models and metamodels for the DSL execution. Within each scenario model, participants could add workload models containing performance information such as test duration and number of virtual users.
  - Script Modeling: In this task, the constructed OpenMLPerf Performance Script Model would simulate the interaction between the user and the system.

In Figure 24, the results obtained from the survey conducted for knowledge assessment are presented. Participants responded to this survey before the evaluation. It is noticeable that all participants mentioned having regular knowledge of UML, and the majority stated having low knowledge about DSL, as well as in modeling PT using UML or other modeling languages. Most participants reported having regular knowledge about PT.

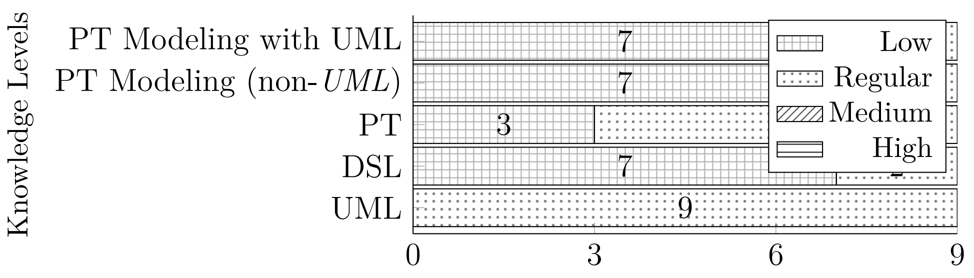
It is important to mention that out of the nine participants who responded to the knowledge assessment survey, signed the participation agreement, and completed the training session, only 8 participants took part in the evaluation, as one of the participants

Approach	Group	Average Time (MM:SS)	Median Time (MM:SS)
UML	1	63:15	63:15
OpenMLPerf	2	66:48	65:30

Table 2 – Effort of Participants (Average and Median)

was absent during the evaluation session. Also, one of the participants requested to be withdrawn from the experiment, even though they had completed the modeling task using the UML profile. The participant did not feel confident enough with their knowledge to share their models for the evaluation, and thus, this participant was excluded from the study. Another participant was excluded due to technical issues with their equipment, which resulted in corruption of the models they had created using the UML profile. The effort values from these two excluded participants were not included in the experiment's results.

Figure 24 – Results of the leveling questions



6.3.3 Results

In this section, the results obtained during the execution of the empirical evaluation will be presented and discussed.

RQ1. *What is the effort required to model performance tests using a UML profile and OpenMLPerf?*

Table 2 presents a summary of the data related to the effort of each participant in performing the modeling task, as described in Section 6.3.2. In the column "Average Time," the effort, measured as the average time, taken by each participant to complete the tasks can be observed. For the UML approach, participants took an average of 1h02min56s to model the proposed scenario and script for the system. This time is shorter compared to the average time of 1h06min40s taken by participants to complete the task using OpenMLPerf.

In Table 3, the individual effort of each participant in performing the modeling task is presented. It can be observed that there is a close proximity between the modeling times using both approaches. Figure 6.3.3 provides a statistical summary of the datasets. The median execution time for the UML approach was 63.5 minutes, which is lower than

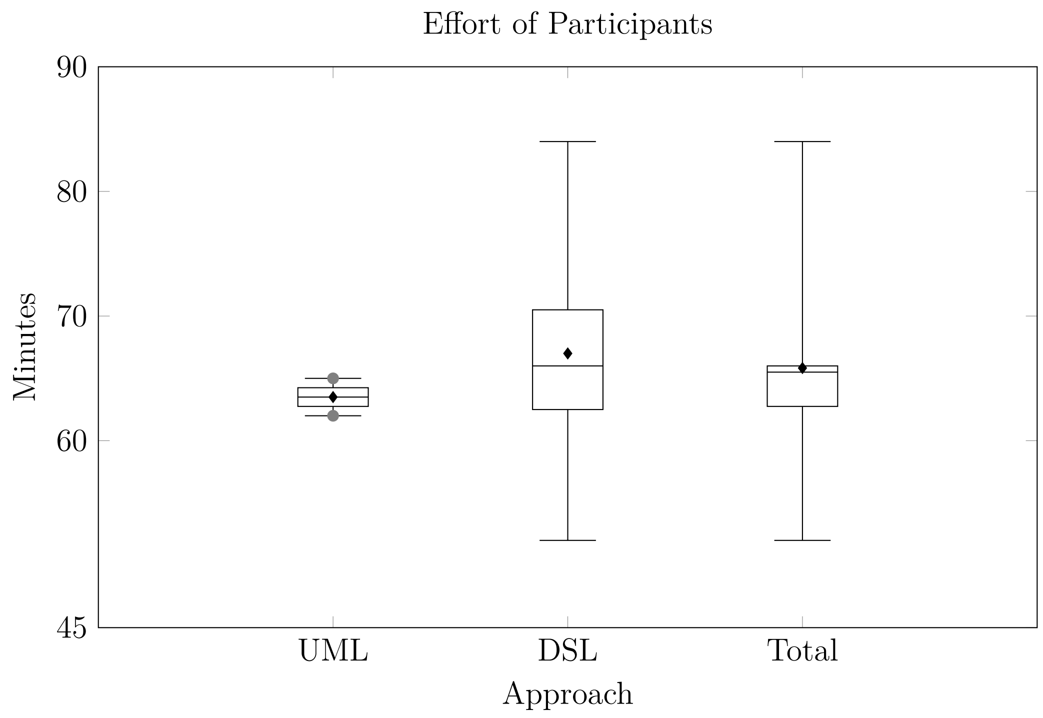


Figure 25 – Effort of participants

the median time for the OpenMLPerf approach, which was 66 minutes. The UML approach also had a lower standard deviation of 2.12 minutes compared to the OpenMLPerf approach, which had a standard deviation of 13.11 minutes.

Table 3 – Effort per participant

Participant	Approach	Time (MM:SS)
P01	UML	64:01
P03	DSL	61:50
P05	UML	65:32
P06	DSL	51:10
P07	DSL	65:32
P08	DSL	84:32

Since there was no sufficiently large sample size, it was not possible to perform normalization and hypothesis testing. Therefore, we considered only the basic statistical values presented in the results.

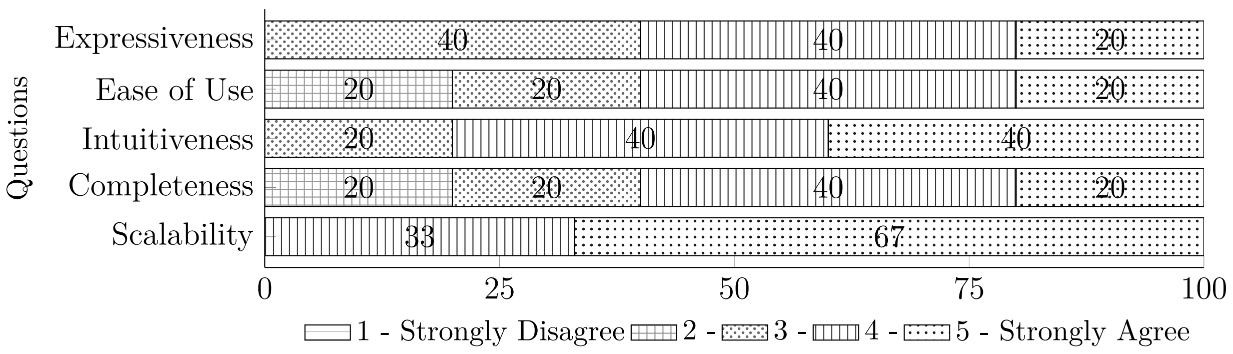
RQ2. *How effective is it to model a performance test using a UML profile and OpenMLPerf?* **RQ3.** *How intuitive/easy is it to model performance tests using a UML profile and OpenMLPerf?*

To answer these two questions, participants were asked to complete a survey after the evaluation, where they were questioned about concepts related to *expressiveness, ease of use, intuitiveness, completeness, scalability.* Conceptually, these terms can be defined as follows: *Expressiveness* - The ability of the DSL to adequately express functional and non-functional requirements related to the system and performance; *Ease of use* - The ease and clarity of modeling performance tests using the DSL; *Completeness* - The ability to model test scenarios with the DSL without the need for additional support materials; *Scalability* - Whether it is possible to increase the complexity of the modeled scenarios. Participants also answered open-ended questions regarding modeling with OpenMLPerf and provided suggestions for improving the experiment.

The questions were structured to be answered using the Likert scale for questionnaire responses. After each question, participants were asked to choose one of the following statements: ***1 - Strongly Disagree, 2 - Disagree, 3 - Neither Agree nor Disagree, 4 - Agree, 5 - Strongly Agree***.

Figure 26 shows the data obtained from the survey. Based on the compiled responses, we can affirm that: 60% (40% - Agree, 20% - Strongly Agree) of participants consider that the DSL is expressive, and likewise, 60% (40% - Agree, 20% - Strongly Agree) of participants found it easy to use the DSL. Regarding intuitiveness, 80% (40% - Agree, 40% - Strongly Agree) of respondents found OpenMLPerf intuitive for modeling. 60% (40% - Agree, 20% - Strongly Agree) of respondents did not feel the need to consult documentation in order to perform modeling with the DSL. Meanwhile, 100% of participants perceived a good level of scalability during the modeling process (33% - Agree, 67% - Strongly Agree).

Figure 26 – Results of questions about OpenMLPerf



The Figure 27 presents the result of the question asking whether the participant would recommend the use of OpenMLPerf to a colleague or manager. 80

Among the benefits of using the OpenMLPerf approach, participants mentioned ease of use, intuitiveness, and practicality when inserting parameters, as it was not necessary to enter tagged values for each parameter and performance property. As a disadvantage, participants mentioned some usability issues when defining parameters. Some suggestions for improvement were also mentioned, such as changing the images that represent the initial and final activities in the activity diagram, which is the representation of the scripts. Participants also reported performance issues when running the DSL in the Eclipse development environment. Some participants found the task a bit challenging and suggested simpler and more concise scenarios and use cases.

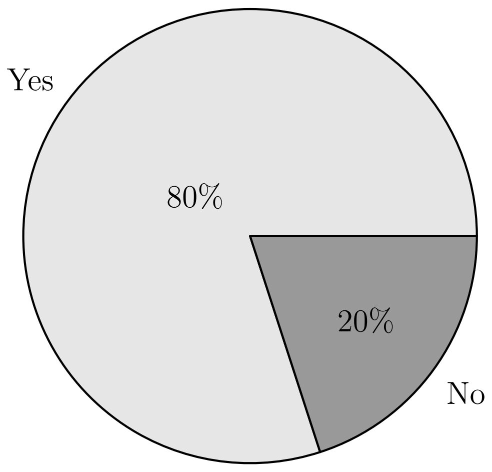


Figure 27 – Results regarding the recommendation of using the DSL

6.3.4 Discussion

In this section, we will discuss the results obtained regarding OpenMLPerf and also the execution of the evaluation, as well as improvements that can be applied to enhance a future execution of the evaluation.

There is a noticeable similarity between the results obtained with modeling using both approaches, as there is no significant difference in the measured effort between the UML profile and OpenMLPerf. When analyzing the data obtained from the evaluation, we can observe that the modeling effort was lower with the UML profile compared to modeling with OpenMLPerf. However, we must take into consideration certain factors that may have influenced the results. One example is the reported issues such as performance drops, freezing, and very slow loading screens in the Eclipse development environment. Although this problem can be mitigated in an empirical evaluation by using controlled environments and standardized machines, it indicates that for individual use or in corporate environments that do not have adequate machines, the Eclipse environment may not be the best option for using the developed language. Another contributing factor is the participants' familiarity with the development environment of the profile and UML itself.

As seen in Table 4, the lack of uniformity in the effort results with the OpenMLPerf approach is also noteworthy. The standard deviation was almost 13 times higher between the two approaches (13.58096094 - DSL, compared to 1.767766953 - UML profile). As an explanation for this, we can mention a few factors, such as the participant with the longest time, who experienced serious performance issues with the Eclipse environment and had no prior experience with DSLs and performance testing. On the other hand, the

Approach	Group	Standard Deviation
UML	1	1.767766953
OpenMLPerf	2	13.58096094

Table 4 – Standard deviation for participant effort

participant with the shortest time had regular knowledge in both DSLs and performance testing.

Another aspect to be addressed in a future evaluation is the execution of multiple sessions or tasks, which would allow for the cross-referencing of results from the execution of different approaches. This would enable one group to use both approaches and compare the modeling of different aspects of the language, such as scenarios and scripts. Model verification could also be performed to identify modeling errors, including syntax errors and activity modeling. Additionally, as suggested by the participants, simplifying and compacting the scenarios and requirements would help maintain participants' focus and optimize the available time for task completion. To mitigate the effects of the learning curve, it would be beneficial to increase the time and training sessions for both approaches and performance testing concepts, as it is unlikely that we will have experts in this area available again.

As part of the ongoing development of the language, a new evaluation is planned to be conducted in the future with computer science and software engineering courses at UNIPAMPA.

6.3.5 Chapter Lessons

In this chapter, an evaluation of OpenMLPerf was presented, focusing on factors such as effort, effectiveness, and ease of use. Through the answers to the research questions, improvements to be implemented in both the DSL and future experiments were identified. Regarding the DSL, aspects such as graphical elements representing performance testing domain objects, parameter creation, and performance-related questions of the LW that impact the execution of the language were observed in the obtained results. The experiment's execution allowed us to verify the importance of participants' profiles and the impact of this aspect on the results. In the next chapter, the final considerations of this work will be presented, along with the lessons learned during the development and evaluation of OpenMLPerf. Future work and research directions will also be discussed.

7 CONCLUSION

In this chapter, we will discuss the final considerations of this work. Furthermore, we will explore future work possibilities.

Testing complex systems and having the necessary knowledge of the technologies required for these tests can be a costly task for a software team. Thus, having a team member who understands and is capable of modeling the domain and the system, regardless of the technology used at a lower layer in the testing activity, is an attractive alternative to address testing-related issues. MDE activities, such as MBT, using DSLs, emerge as viable and practical options for testing large-scale systems, both established and new, during the conception phase.

In this work, we presented the implementation of a DSL for performance testing modeling of web systems, OpenMLPerf. After discussing the concepts related to the domain of specific modeling languages, we presented the requirements and design decisions for the language project. A brief description of a systematic literature review, which was performed to assist in choosing the most appropriate tools for language implementation, was also presented and detailed in this work. Finally, an empirical evaluation was conducted to quantify the user's effort in modeling with the DSL, allowing a comparison with performance testing modeling using a UML profile.

7.1 Lessons Learned

Some challenges were encountered in choosing suitable technologies and learning about their proper use during the course of this work. The execution of the SMS helped, among other things, to understand the DSLs DSL and the functionalities offered by Language Workbenches (LWs). Thus, limiting the number of suitable tools and deciding which one would best fit the research needs became a less costly activity in terms of implementation effort. However, it does not mean that the implementation itself became easier.

Working with cutting-edge technologies, which are often still in development and lack adequate or even non-existent support, is one of the burdens of trying to implement innovative and relevant systems. In this work, we faced the challenge of establishing concepts related to the necessary syntax for building the language. This includes metametamodeling elements such as Ecore, as well as modeling elements like UML. Thus, it was necessary to remodel the entire performance testing domain since the original metamodels of Canopus could not be reused, as MetaEdit+ uses the concept of GOPPRR (Graph, Object, Port, Property, Relationship, and Role), while Ecore has its different concept, based on classes and relationships, just like UML. As a result, all the metamodels had to be redeveloped.

The implementation of the language with the Sirius framework cannot be considered a difficult task since the purpose of LWs is to facilitate this activity, but it cannot be considered trivial either, mainly due to the limitations of the LW itself, which is constantly evolving. Thus, much of the difficulty encountered results from the learning curves and lack of support for certain features, such as dynamically creating models generated through resources available in other models (e.g., metric model allocated in Systems Under Test). There are also common limitations in Eclipse environments, such as slow loading, errors, and exceptions.

The empirical evaluation conducted in this work, despite its limitations, cannot be discarded, as it serves as a basis and can be considered a trial or pilot for a new execution of the evaluation. This would allow for the correction of errors found and improvement in the generality of samples, as well as detecting limitations beyond the control of the developer, such as LWs performance and usability limitations. However, it is important to mention that only the modeling functionality was put under evaluation, while the code generation will be evaluated in future works.

7.2 Future Work

One of the main activities to be carried out in the future is the execution of a new empirical evaluation of OpenMLPerf. This should aim to ensure a larger and more diverse number of participants, thus allowing for a more robust, accurate, and satisfactory statistical analysis. Another evaluation activity to be conducted is the assessment of code generation in the language, enabling a comparison with manual code writing.

Regarding the language itself, it is expected that in the future, it can be integrated with a textual notation for performance testing, using the Xtext framework and generating code in both notations bidirectionally.

BIBLIOGRAPHY

ABBORS, F. et al. Mbpet: a model-based performance testing tool. In: **2012 Fourth International Conference on Advances in System Testing and Validation Lifecycle**. [S.l.: s.n.], 2012. Citated in page 35.

All4Tec. **Matelo**. 2019. Disponível em: <https://all4tec.com>. Citated in page 28.

Apache Foundation. **Apache JMeter**. 2023. Available in: <https://jmeter.apache.org/>. Citated in page 22.

Austrian Institute of Technology. **MoMut**. 2019. Disponível em: <https://momut.org/>. Citated in page 28.

BERNARDINO, M. Canopus: a domain-specific language for modeling performance testing. Pontifícia Universidade Católica do Rio Grande do Sul, 2016. Citated 3 time in the pages 21, 35, and 47.

BERNARDINO, M.; ZORZO, A. F.; RODRIGUES, E. M. Canopus: A Domain-Specific Language for Modeling Performance Testing. In: **2016 IEEE International Conference on Software Testing, Verification and Validation (ICST'16)**. [S.l.: s.n.], 2016. p. 157–167. Citated in page 34.

BUDINSKY, F. et al. **Eclipse modeling framework: a developer's guide**. [S.l.]: Addison-Wesley Professional, 2004. Citated in page 31.

ChangeVision. **Astah Modeler**. 2023. Disponível em: <https://astah.net/>. Citated in page 70.

Eclipse Foundation. **Ecore model description**. 2018. Available in: <http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html#details>. Citated in page 32.

Eclipse Foundation. **Framework XText**. 2018. Available in: <https://www.eclipse.org/Xtext/>. Citated in page 44.

Eclipse Foundation. **Acceleo**. 2023. Https://eclipse.dev/acceleo/overview.html. Citated 2 time in the pages 32 and 33.

Eclipse GEMOC Research Consortium. **GEMOC**. 2023. Available in: https://gemoc.org/. Citated in page 44.

EL-FAR, I. K.; WHITTAKER, J. A. Model-based software testing. **Encyclopedia of Software Engineering**, Wiley Online Library, 2002. Citated in page 27.

ERDWEG, S. et al. The state of the art in language workbenches. In: **Software Language Engineering**. Cham: Springer International Publishing, 2013. ((SLE'13)), p. 197–217. ISBN 978-3-319-02654-1. Citated in page 44.

ERDWEG, S. et al. Evaluating and comparing language workbenches: Existing results and benchmarks for the future. **Computer Languages, Systems & Structures**, Elsevier, v. 44, p. 24–47, 2015. Citated in page 44.

FERME, V.; PAUTASSO, C. A declarative approach for performance tests execution in continuous software development environments. In: ACM. **Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering**. [S.l.], 2018. p. 261–272. Citated in page 36.

FOWLER, M. **Domain-specific languages**. [S.l.]: Pearson Education, 2010. Citated 2 time in the pages 21 and 29.

FRANCE, R.; RUMPE, B. Model-driven development of complex software: A research roadmap. In: IEEE COMPUTER SOCIETY. **2007 Future of Software Engineering**. [S.l.], 2007. p. 37–54. Citated 2 time in the pages 21 and 27.

FUENTES-FERNÁNDEZ, L.; VALLECILLO-MORENO, A. An introduction to uml profiles. **UML and Model Engineering**, v. 2, 2004. Citated in page 29.

Gatling. **Gatling Stress Tool**. 2018. Disponível em: <http://gatling.io/>. Citated in page 37.

Hewlett Packard. **Software HP LoadRunner**. 2018. Disponível em: <https://software.microfocus.com/pt-br/products/loadrunner-load-testing/free-trial>. Citated 2 time in the pages 21 and 37.

HYMAN, R. Quasi-experimentation: Design and analysis issues for field settings (book). **Journal of Personality Assessment**, Taylor & Francis, v. 46, n. 1, p. 96–97, 1982. Citated in page 70.

IUNG, A. et al. Systematic mapping study on domain-specific language development tools. **Empirical Software Engineering**, Springer, v. 25, p. 4205–4249, 2020. Citated 2 time in the pages 39 and 45.

JetBrains. **Meta Programming System**. 2023. Available in: <https://www.jetbrains.com/mps/>. Citated in page 44.

JOHNSON, R. E. Frameworks = (components + patterns). **Communications of ACM**, ACM, New York, NY, USA, v. 40, n. 10, p. 39–42, out. 1997. ISSN 0001-0782. Citated in page 29.

KELLY, S.; TOLVANEN, J.-P. **Domain-Specific Modeling: Enabling Full Code Generation**. New York, NY, USA: John Wiley & Sons, 2007. ISBN 0470036664. Citated in page 29.

KENT, S. Model driven engineering. In: SPRINGER. **International Conference on Integrated Formal Methods**. [S.l.], 2002. p. 286–298. Citated in page 27.

KORENKOV, Y.; LOGINOV, I.; LAZDIN, A. Peg-based language workbench. In: **Open Innovations Association (FRUCT), 2015 17th Conference of**. Yaroslavl, Russia: [s.n.], 2015. v. 2015-June, p. 75 – 81. ISSN 23057254. Citated in page 29.

LE, D. M.; DANG, D.-H.; NGUYEN, V.-H. On domain driven design using annotation-based domain specific language. **Computer Languages, Systems & Structures**, Elsevier, 2018. Citated in page 21.

MARK, K.; CSABA, L. Analyzing customer behavior model graph (cbmg) using markov chains. In: **2007 11th International Conference on Intelligent Engineering Systems**. [S.l.: s.n.], 2007. p. 71–76. Citated in page 35.

MEIER, J. et al. **Performance testing guidance for web applications: patterns & practices**. [S.l.]: Microsoft press, 2007. Citated in page 28.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. **ACM computing surveys (CSUR)**, ACM, v. 37, n. 4, p. 316–344, 2005. Citated in page 29.

MetaCase. **MetaEdit+ Modeler**. 2018. Disponível em: <https://www.metacase.com/mep/>. Citated 3 time in the pages 22, 34, and 44.

MOLYNEAUX, I. **The art of application performance testing: from strategy to tools**. [S.l.]: " O'Reilly Media, Inc.", 2014. Citated in page 21.

NAKAMURA, H. et al. Qoral: An external domain-specific language for mining software repositories. In: **Proceedings of the 2012 Fourth International Workshop on Empirical Software Engineering in Practice**. Washington, DC, USA: IEEE Computer Society, 2012. ((IWESEP'12)), p. 23–29. ISBN 978-0-7695-4866-1. Citated in page 35.

PEREZ, F.; VALDERAS, P.; FONS, J. A domain-specific language for enabling doctors to specify biomechanical protocols. In: **2013 IEEE Symposium on Visual Languages and Human Centric Computing**. [S.l.: s.n.], 2013. ((IEEE-VL/HCC'13)), p. 99–102. ISSN 1943-6092. Citated in page 35.

PETERSEN, K. et al. Systematic mapping studies in software engineering. **12th Int. Conf. on Evaluation and Assessment in Software Engineering**, British Computer Society, v. 17, n. 1, p. 1–10, 2008. Citated in page 39.

RIBIĆ, S. et al. Redosplat: A readable domain-specific language for timetabling requirements definition. **Computer Languages, Systems & Structures**, Elsevier, v. 54, p. 252–272, 2018. Citated 2 time in the pages 21 and 35.

RODRIGUES, E. et al. Pletsperf-a model-based performance testing tool. In: IEEE. **2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)**. [S.l.], 2015. p. 1–8. Citated in page 67.

RUFFO, G. et al. Walty: a user behavior tailored tool for evaluating web application performance. In: IEEE. **Third IEEE International Symposium on Network Computing and Applications, 2004.(NCA 2004). Proceedings.** [S.l.], 2004. p. 77–86. Citated 2 time in the pages 35 and 36.

SMITH, C. U.; WILLIAMS, L. G. Software performance engineering: A case study including performance comparison with design alternatives. **IEEE Transactions on software engineering**, IEEE, v. 19, n. 7, p. 720–741, 1993. Citated in page 28.

SPAFFORD, K. L.; VETTER, J. S. Aspen: a domain specific language for performance modeling. In: IEEE COMPUTER SOCIETY PRESS. **Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis**. [S.l.], 2012. p. 84. Citated in page 37.

STEINBERG, D. et al. **EMF: eclipse modeling framework**. [S.l.]: Pearson Education, 2008. Citated in page 31.

SUN, Y.; WHITE, J.; EADE, S. A model-based system to automate cloud resource allocation and optimization. In: SPRINGER. **International Conference on Model Driven Engineering Languages and Systems**. [S.l.], 2014. p. 18–34. Citated 2 time in the pages 36 and 37.

TANG, W. Meta object facility. In: ______. **Encyclopedia of Database Systems**. Boston, MA: Springer US, 2009. p. 1722–1723. ISBN 978-0-387-39940-9. Citated in page 30.

Test Optimal. **Test Optimal**. 2019. Disponível em: <http://mbt.testoptimal.com/index.html>. Citated in page 28.

UTTING, M.; LEGEARD, B. **Practical model-based testing: a tools approach**. [S.l.]: Elsevier, 2010. Citated in page 27.

VIYOVIĆ, V.; MAKSIMOVIĆ, M.; PERIŠIĆ, B. Sirius: A rapid development of DSM graphical editor. **IEEE 18th International Conference on Intelligent Engineering Systems, Proceedings (INES'14)**, p. 233–238, 2014. Citated in page 31.

WACHSMUTH, G. H.; KONAT, G. D. P.; VISSER, E. Language design with the spoofax language workbench. **IEEE Software**, v. 31, n. 5, p. 35–43, Sept 2014. ISSN 0740-7459. Citated in page 29.

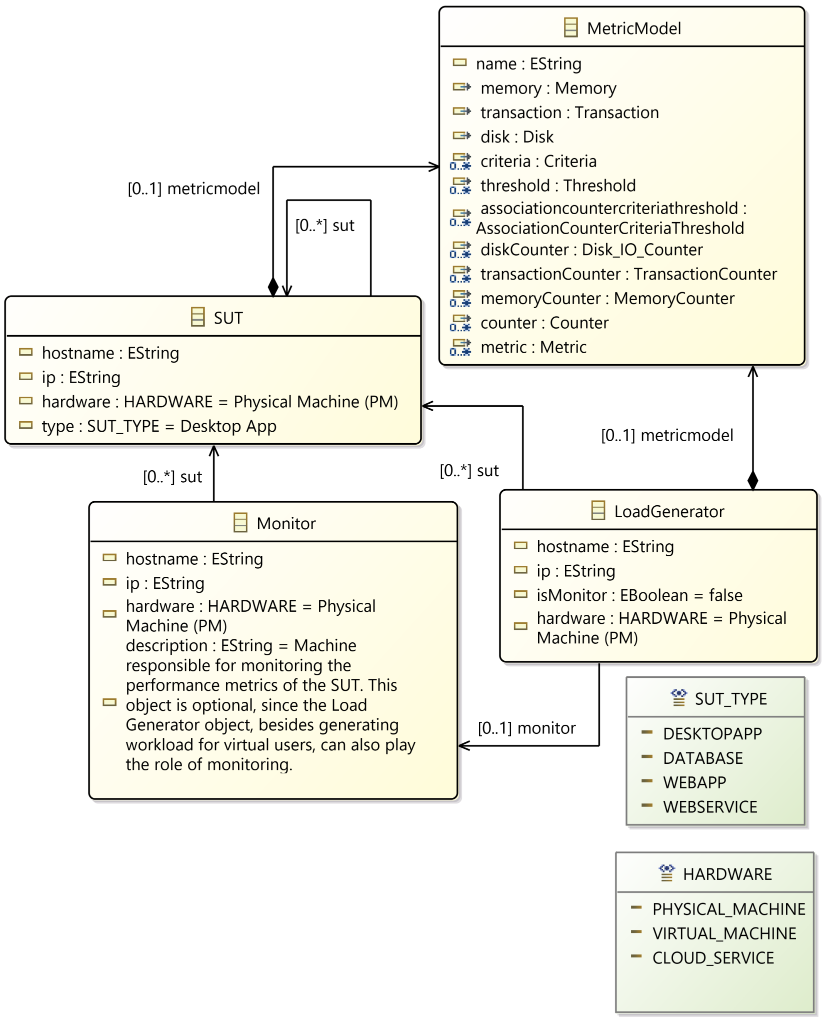
WACHSMUTH, G. H.; KONAT, G. D. P.; VISSER, E. Language design with the spoofax language workbench. **IEEE Software**, v. 31, n. 5, p. 35–43, 2014. ISSN 07407459. Citated in page 44.

ZHAO, T.; HUANG, X. Design and implementation of deepdsl: A dsl for deep learning. **Computer Languages, Systems & Structures**, Elsevier, v. 54, p. 39–70, 2018. Citated in page 21.

Appendix

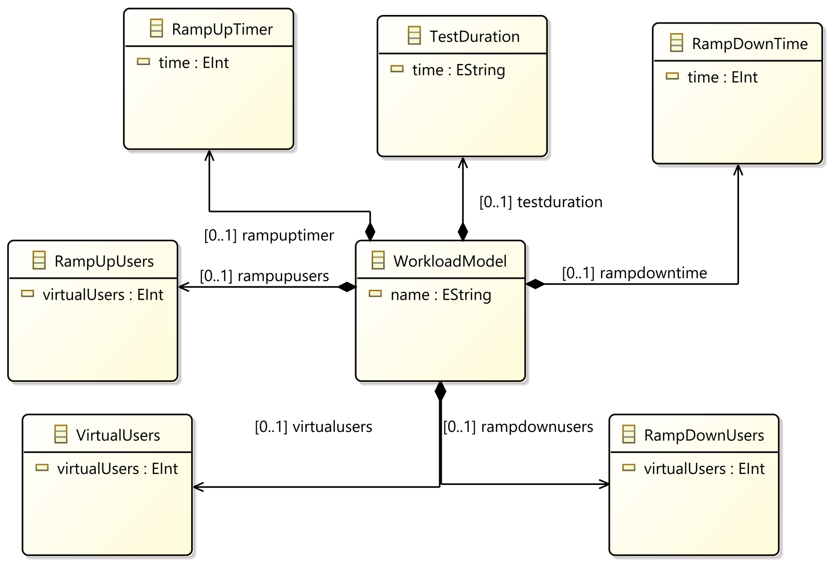
APPENDIX A – APÊNDICE A

Figure 28 – Class diagram displaying the monitoring metamodel of the language


MetricModel
name : EString
memory : Memory
transaction : Transaction
disk : Disk
criteria : Criteria
threshold : Threshold
associationcountercriteriathreshold : AssociationCounterCriteriaThreshold
diskCounter : Disk_IO_Counter
transactionCounter : TransactionCounter
memoryCounter : MemoryCounter
counter : Counter
metric : Metric
[0..1] metricmodel
[0..*] sut
SUT
hostname : EString
ip : EString
hardware : HARDWARE = Physical Machine (PM)
type : SUT_TYPE = Desktop App
[0..1] metricmodel
[0..*] sut
[0..*] sut
Monitor
hostname : EString
ip : EString
hardware : HARDWARE = Physical Machine (PM)
description : EString = 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.
LoadGenerator
hostname : EString
ip : EString
isMonitor : EBoolean = false
hardware : HARDWARE = Physical Machine (PM)
[0..1] monitor
SUT_TYPE
DESKTOPAPP
DATABASE
WEBAPP
WEBSERVICE
HARDWARE
PHYSICAL_MACHINE
VIRTUAL_MACHINE
CLOUD_SERVICE

Source: Author

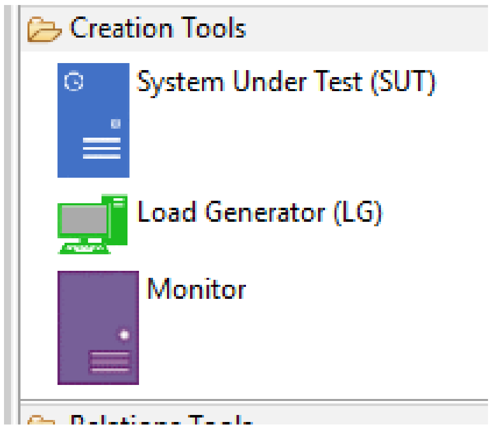
Figure 29 – Class diagram showing the metamodel of the workload



Source: Autor

APPENDIX B –

Figure 30 – Modeling diagram of the language's scripts



Source: Autor

APPENDIX C –

```
package scenariolabel

import scala.concurrent.duration._

import io.gatling.core.Predef._
import io.gatling.http.Predef._
import io.gatling.jdbc.Predef._

class Scenario1 extends Simulation{



        object Edit {



val feederAllCategoriescsv ("AllCategories.csv").random






val feederAllBookscsv ("AllBooks.csv").random






        val edit = scenario("RecordedSimulation")




                        .exec(http("request_0")
                        .get("Testset")
```

```
                    .exec(http("request_0")
                    .get("")




                    .exec(http("request_0")
                    .get("")

                    .feed(feederAllCategoriescsv)




                    .exec(http("request_0")
                    .get("")

                    .feed(feederAllCategoriescsv)




                    .exec(http("request_0")
                    .get("")

                    .feed(feederAllBookscsv)




                    .exec(http("request_0")
                    .get("")






          }
```

```
        object Browser {


val feederSubjectcsv ("Subject.csv").random




val feederBooksByCategorycsv ("BooksByCategory.csv").random




val feederSearchResultscsv ("SearchResults.csv").random




        val browser = scenario("RecordedSimulation")




                .exec(http("request_0")
                .get("")





                .exec(http("request_0")
                .get("")

                .feed(feederSubjectcsv)
```

```
                    .exec(http("request_0")
                    .get("")

                    .feed(feederSubjectcsv)





                    .exec(http("request_0")
                    .get("")

                    .feed(feederBooksByCategorycsv)





                    .exec(http("request_0")
                    .get("")






                    .exec(http("request_0")
                    .get("")

                    .feed(feederSearchResultscsv)






    }

    //Shopping 25%
    //Browsing 15%

    //Load Testing : 10000 virtual users
          0
          5000
    //Endurance Testing : 2000 virtual users
```

```
                2000
                500

val httpProtocol = http
                .baseUrl("")
                .inferHtmlResources()
                .acceptEncodingHeader("gzip, deflate")
                .acceptLanguageHeader("pt-BR,pt;q=0.9,en-US;q=
                   0.8,en;q=0.7,sv;q=0.6")
                .userAgentHeader("Mozilla/5.0 (Windows NT 10.0;
                   Win64; x64) AppleWebKit/537.36 (KHTML, like
                   Gecko) Chrome/84.0.4147.125 Safari/537.36")

val headers_0 = Map(
                "Accept" -> "text/html,application/xhtml+xml,
                   application/xml;q=0.9,image/webp,image/apng
                   ,*/*;q=0.8,application/signed-exchange;v=b3;q=
                   0.9",
                "Sec-Fetch-Dest" -> "document",
                "Sec-Fetch-Mode" -> "navigate",
                "Sec-Fetch-Site" -> "none",
                "Sec-Fetch-User" -> "?1",
                "Upgrade-Insecure-Requests" -> "1")



}
```

Listing C.1 – Code snippet generated by the OpenMLPerf

APPENDIX D –

Questionário Pré-Experimento

Introdução
Gostaríamos de convidá-lo a participar de um estudo referente a pesquisa de Linguagens Específica de Domínio (DSL) para modelagem de teste de desempenho. Este estudo irá prover uma avaliação empírica para nossa DSL.

Por favor leia este formulário e pergunte qualquer questão que você possa ter antes de concordar em participar deste estudo.

Este estudo está sendo conduzido por: João Batista Pedroso Carbonell, graduando no curso de Engenharia de Software da Unipampa, orientado pelo Prof. Dr. Elder de Macedo Rodrigues, com a colaboração do Prof. Dr. Fabio Paulo Basso e Prof Dr. Maicon Bernardino da Silveira, professores na Unipampa.

Contexto: O objetivo deste survey é avaliar o perfil do entrevistado para mapear e randomizar os participantes do experimento entre os tratamentos do experimento.

Procedimentos: Se você aceitar participar neste estudo, será convidado a responder questões a respeito de seu conhecimento e habilidades em aspectos técnicos. As questões são de múltipla escolha. Deve levar entre 5-10 minutos para completar o survey.

Riscos: Ser um participante neste estudo na possui riscos previsíveis.

Benefícios: O pesquisador espera avaliar a DSL para modelagem de testes de desempenho comparando-a uma abordagem baseada em UML. Isto pode ajudar a profissionais de teste de desempenho a realizarem modelagens de cenários e scripts mais eficientes. Como um participante, você pode ter acesso ao resultado desta pesquisa.

Confidencialidade: As informações e arquivos deste estudo serão mantidos em privado. Qualquer tipo de publicação não deve conter qualquer informação que possibilite a identificação de um participante. Apenas os pesquisadores primários deverão ter acesso aos arquivos.

Contato e perguntas: O pesquisador condutor desta pesquisa é João Cabonell. Por favor, contate-o com qualquer pergunta.

joaocarbonellpc@gmail.com

Se você deseja participar, por favor inicie o survey respondendo a seguinte questão:

*Obrigatório

1. **Endereço de e-mail** *

2. **Escreva seu nome** *

3. **Escreva o nome da instituição onde realizou ou realiza a graduação** *

4. **Qual o curso de graduação?** *

Marcar apenas uma oval.

- Engenharia de Software
- Ciência da Computação
- Engenharia da Computação
- Sistemas da Informação
- Analise e Desenvolvimento de Sistemas
- Outro: ______________________

5. **Quantos anos de experiencia você tem em engenharia de software?** *

Marcar apenas uma oval.

- 14+
- 11 -- 13
- 8 -- 10
- 5 -- 7
- 2 -- 4
- 0 -- 1 ano

6. **Como você classifica seu conhecimento técnico em modelagem de software com UML?** *

Marcar apenas uma oval.

- Baixo, sem conhecimento prévio
- Regular, leitura de livro ou realizado algum curso
- Médio, alguma experiencia industrial (menos de 6 meses)
- Alto, experiencia industrial

7. **Como você classifica seu conhecimento em Linguagens Específicas de Domínio (DSL)?** *

Marcar apenas uma oval.

- Baixo, sem conhecimento prévio
- Regular, leitura de livro ou realizado algum curso
- Médio, alguma experiencia industrial (menos de 6 meses)
- Alto, experiencia industrial

8. **Como você classifica seu conhecimento em Teste de Desempenho?** *

Marcar apenas uma oval.

- Baixo, sem conhecimento prévio
- Regular, leitura de livro ou realizado algum curso
- Médio, alguma experiencia industrial (menos de 6 meses)
- Alto, experiencia industrial

9. **Como você classifica seu conhecimento em modelagem de teste de desempenho com notações ou linguagens de modelagem?** *

Marcar apenas uma oval.

- ◯ Baixo, sem conhecimento prévio
- ◯ Regular, leitura de livro ou realizado algum curso
- ◯ Médio, alguma experiencia industrial (menos de 6 meses)
- ◯ Alto, experiencia industrial

10. **Como você classifica seu conhecimento em modelagem de teste de desempenho com UML?** *

Marcar apenas uma oval.

- ◯ Baixo, sem conhecimento prévio
- ◯ Regular, leitura de livro ou realizado algum curso
- ◯ Médio, alguma experiencia industrial (menos de 6 meses)
- ◯ Alto, experiencia industrial

Autorização

Eu concordo em participar neste experimento nas condições que foram propostas. Eu garanto que irei realizar este experimento da melhor maneira que eu puder, garantindo que todas as informações incluídas aqui são reais.

11. **Por favor, escolha uma das seguintes respostas:** *

Marcar apenas uma oval.

- ◯ Concordo
- ◯ Não Concordo

Pós-Experimento DSL Canopus

Introdução
Gostaríamos de convidá-lo a participar de um estudo referente a pesquisa de Linguagens Específica de Domínio (DSL) para modelagem de teste de desempenho. Este estudo irá prover uma avaliação empírica para nossa DSL.

Por favor leia este formulário e pergunte qualquer questão que você possa ter antes de concordar em participar deste estudo.

Este estudo está sendo conduzido por: João Batista Pedroso Carbonell, graduando no curso de Engenharia de Software da Unipampa, orientado pelo Prof. Dr. Elder de Macedo Rodrigues, com a colaboração do Prof. Dr. Fabio Paulo Basso e Prof Dr. Maicon Bernardino da Silveira, professores na Unipampa.

Contexto: O objetivo deste survey é avaliar a percepção sobre o experimento empírico conduzido por ambas as abordagens: DSL e UML para modelagem de teste de desempenho.

Procedimentos: Se você concordar em participar deste estudo, será solicitado a você que complete um questionário destinado a responder 2 questões de pesquisa: 1) O quanto é efetivo construir modelos de teste de desempenho quando utilizando UML ou DSL Canopus? 2) O quão intuitivo/fácil é construir modelos de teste de desempenho quando utilizando UML ou DSL Canopus? A maioria das questões são de múltipla escolha, usando a escala Likert. Entretanto, há um outro conjunto de questões que são respondidas em forma de redação, por exemplo, sugestões ou comentários sobre as vantagens e desvantagens, sob o seu ponto de vista sobre cada uma delas. Isto levará em torno de 15-30 minutos para completar suas questões.

Se você aceitar participar neste estudo, será convidado a responder questões a respeito de seu conhecimento e habilidades em aspectos técnicos. As questões são de múltipla escolha. Deve levar entre 5-10 minutos para completar o survey.

Riscos: Ser um participante neste estudo na possui riscos previsíveis.

Benefícios: O pesquisador espera avaliar a DSL para modelagem de testes de desempenho comparando-a uma abordagem baseada em UML. Isto pode ajudar a profissionais de teste de desempenho a realizarem modelagens de cenários e scripts mais eficientes. Como um participante, você pode ter acesso ao resultado desta pesquisa.

Confidencialidade: As informações e arquivos deste estudo serão mantidos em privado. Qualquer tipo de publicação não deve conter qualquer informação que possibilite a identificação de um participante. Apenas os pesquisadores primários deverão ter acesso aos arquivos.

Contato e perguntas: O pesquisador condutor desta pesquisa é João Cabonell. Por favor, contate-o com qualquer pergunta.

joaocarbonellpc@gmail.com

Se você deseja participar, por favor inicie o survey respondendo a seguinte questão:

Existe(m) 14 questão(ões) neste questionário.

Nome do participante: *

Por favor, coloque sua resposta aqui:

Ao modelar com perfil UML para teste de desempenho, a linguagem forneceu todos os elementos necessários à modelagem de cenários e scripts de teste de desempenho. *

Favor escolher apenas uma das opções a seguir:

- 1
- 2
- 3
- 4
- 5

1 - Discordo fortemente, 2 - Discordo, 3 - Nem concordo ou discordo, 4 - Concordo, 5 - Concordo fortemente

Os modelos de teste de desempenho criados com a abordagem utilizando a DSL Canopus expressam mais adequadamente os requisitos funcionais e não funcionais. *

Favor escolher apenas uma das opções a seguir:

- 1
- 2
- 3
- 4
- 5

1 - Discordo fortemente, 2 - Discordo, 3 - Nem concordo ou discordo, 4 - Concordo, 5 - Concordo fortemente

É mais fácil modelar o teste de desempenho aplicando a abordagem com a DSL Canopus do que aplicando a abordagem com o perfil UML para teste de desempenho. *

Favor escolher apenas uma das opções a seguir:

- 1
- 2
- 3
- 4
- 5

1 - Discordo fortemente, 2 - Discordo, 3 - Nem concordo ou discordo, 4 - Concordo, 5 - Concordo fortemente

É mais intuitivo aplicar a representação do domínio de teste de desempenho utilizando a abordagem com a DSL Canopus do que utilizando a abordagem com o perfil UML para teste de desempenho. *

Favor escolher apenas uma das opções a seguir:

- 1
- 2
- 3
- 4
- 5

1 - Discordo fortemente, 2 - Discordo, 3 - Nem concordo ou discordo, 4 - Concordo, 5 - Concordo fortemente

A abordagem com o perfil UML para teste de desempenho descreve todas as informações necessárias para a modelagem dos cenários de teste, sem consultar outras fontes (*i.e. a* documentação de suporte da referência). *

Favor escolher apenas uma das opções a seguir:

- 1
- 2
- 3
- 4
- 5

1 - Discordo fortemente, 2 - Discordo, 3 - Nem concordo ou discordo, 4 - Concordo, 5 - Concordo fortemente

A abordagem com a DSL Canopus descreve todas as informações necessárias para a modelagem dos cenários de teste, sem consultar outras fontes(*i.e.* a documentação de suporte da referência). *

Favor escolher apenas uma das opções a seguir:

- 1
- 2
- 3
- 4
- 5

1 - Discordo fortemente, 2 - Discordo, 3 - Nem concordo ou discordo, 4 - Concordo, 5 - Concordo fortemente

A abordagem com o perfil UML para teste de desempenho apoia o aumento da complexidade dos cenários de teste (*i.e.* aumentar o teste ou tamanho do modelo, aumentar a quantidade de teste). *

Favor escolher apenas uma das opções a seguir:

- 1
- 2
- 3
- 4
- 5

1 - Discordo fortemente, 2 - Discordo, 3 - Nem concordo ou discordo, 4 - Concordo, 5 - Concordo fortemente

A abordagem com a DSL Canopus apoia o aumento da complexidade dos cenários de teste (*i.e.* aumentar o teste ou tamanho do modelo, aumentar a quantidade de teste). *

Favor escolher apenas uma das opções a seguir:

- 1
- 2
- 3
- 4
- 5

1 - Discordo fortemente, 2 - Discordo, 3 - Nem concordo ou discordo, 4 - Concordo, 5 - Concordo fortemente

Descreva abaixo quais os pontos positivos identificados durante a modelagem de teste de desempenho utilizando a abordagem com perfil UML para teste de desempenho. *

Por favor, coloque sua resposta aqui:

Descreva abaixo quais os pontos positivos identificados durante a modelagem de teste de desempenho utilizando a abordagem com a DSL Canopus. *

Por favor, coloque sua resposta aqui:

Descreva abaixo quais os pontos negativos identificados durante a modelagem de teste de desempenho utilizando a abordagem com a DSL Canopus. *

Por favor, coloque sua resposta aqui:

Você recomendaria a abordagem com a DSL Canopus para a modelagem de teste de desempenho a algum colega ou convenceria um gerente a investir? Se não, porquê? Se sim, argumente o porquê você usaria? *

Escolha uma das seguintes respostas:
Favor escolher apenas uma das opções a seguir:

- Sim
- Não

Comente aqui sua escolha:

Quais sugestões você traz para melhorar a execução deste experimento?

Por favor, coloque sua resposta aqui:

Obrigado pela sua contribuição
26.04.2019 – 23:55

Enviar questionário
Obrigado por ter preenchido o questionário.