UNIVERSIDADE FEDERAL DO PAMPA

João Batista Pedroso Carbonell

OpenMLPerf - Linguagem Específica de Domínio Gráfica Para Testes de Desempenho em Sistemas Web

Alegrete
2019

João Batista Pedroso Carbonell

OpenMLPerf - Linguagem Específica de Domínio Gráfica Para Testes de Desempenho em Sistemas Web

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Orientador: Prof. Dr. Élder de Macedo Rodrigues

Coorientador: Prof. Dr. Maicon Bernardino da Silveira

Alegrete
2019

João Batista Pedroso Carbonell

OpenMLPerf - Linguagem Específica de Domínio Gráfica Para Testes de Desempenho em Sistemas Web

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Trabalho de Conclusão de Curso defendido e aprovado em 28 de Junho de 2017
Banca examinadora:

Prof. Dr. Élder de Macedo Rodrigues
Orientador
UNIPAMPA

Prof. Dr. Maicon Bernardino da Silveira
Coorientador
UNIPAMPA

Prof. Dr. Avelino Francisco Zorzo
PUCRS

Prof. Dr. Fábio Paulo Basso
UNIPAMPA

“A ciência é muito mais que um corpo de conhecimentos. É uma maneira de pensar.

(Carl Sagan)

RESUMO

Testar o desempenho de um sistema é uma importante atividade realizada durante o processo de desenvolvimento de software, assim como, durante a sua evolução e na manutenibilidade das infraestruturas que o cercam. Atualmente, grande parte dos projetos de testes de desempenho consistem em testar um sistema que já foi implementado. Isto torna difícil prever quais problemas precisam ser corrigidos durante o desenvolvimento, tais como carga suportada e gargalos rede. Por estes motivos é recomendável investigar mecanismos e métodos para realizar a modelagem e especificar as informações do sistema, antes e durante o processo de desenvolvimento. Uma das soluções encontradas na literatura é a utilização de modelos que representem aspectos imutáveis do domínio do sistema, como características da infraestrutura ou de comportamentos dos usuários.
Utilizar modelos para abstração, definição e modelagem de aspectos de um domínio vem sendo uma das abordagens mais utilizada atualmente para resolver problemas de sistemas em contextos específicos, o qual vem ao encontro com um dos princípios da Engenharia Dirigida por Modelos (MDE - Model Driven Engineering), que é utilizar modelos em todas as etapas do ciclo de vida do software. Uma das formas de especificar e modelar um domínio é a utilização de Linguagens Específicas de Domínio (*Domain-Specific Language* - DSL). DSLs são pequenas linguagens, com expressividade limitada a um determinado domínio. Diferentemente de linguagens convencionas de propósito geral, como Python e JAVA, DSLs não podem ser utilizadas para programar sistemas para qualquer domínio, sendo restritas apenas ao seu propósito inicial. Este trabalho apresenta a implementação de um DSL gráfica, baseada na DSL Canopus, para a modelagem de testes de desempenho para sistemas web. Na implementação original da Canopus foram utilizadas tecnologias de licença comercial, o que restringiu a sua adoção, distribuição e evolução. Assim, propomos uma nova versão, chamada OpenMLPerf, que será distribuída de forma aberta para uso, melhorias e evolução. Para dar suporte na seleção das ferramentas de suporte ao desenvolvimento de DSL utilizada neste trabalho foi realizado um Mapeamento Sistemático da Literatura, onde foram encontradas 53 ferramentas, que apoiam alguma etapa do processo de desenvolvimento de DSL. Ao final, foram escolhidos os frameworks Ecore, do Eclipse Modeling Framework (EMF) e Sirius para a implementação do metamodelo e linguagem de modelagem. Para avaliar a DSL proposta foi conduzida uma avaliação empírica com a intenção de verificar o esforço dedicado pelo usuário para realizar a modelagem de um cenário de teste de desempenho utilizando a abordagem com a DSL Canopus em comparação com um perfil UML para teste de desempenho. Os resultados apresentam evidências de que ao realizar a modelagem com ambas as abordagens, houve um proximidade nos esforços dos participantes, entretanto, a abordagem com a OpenMLPerf foi prejudicada por problemas de desempenho relacionados a plataforma Eclipse.

Palavras-chave: Linguagem Específica de Domínio. Teses Baseados em Modelos. En-

genharia Dirigida por Modelos. Teste de Desempenho.

ABSTRACT

Test a system performance is one of the most important tasks in software development process and evolution. Most of the performance tests currently applied consist on testing an implemented system, which make difficult to predict problems, such as load capacities and bottlenecks. Thus, it is recommended to find mechanisms for modeling and specifing the system information before the implementation. A possible solution is the use of models which represent unchangeable aspects from the system domain, such as infrastructure features or users behavior.

Use models to abstract, define and model aspects from a domain, nowadays become the most commonly used approaches to solve a problem in a specific context which meets with one of Model-Driven Engineering (MDE) principles, which are to use models in all software cycle-life. One way to specify and model a domain is using Domain-Specific Languages (DSL). DSL are small languages, with limited expressiveness to a determined domain. Different from general purpose languages, such as python and JAVA, DSL are not used to implement systems for any domain, being restricted just to their initial purpose. In this work, we propose the reimplementation of a graphical DSL, named as Canopus, to model performance tests for web systems. In Canopus implementation has used commercial licenses technologies, this restricted its distribution and evolution. Thus, is proposed a new version of this DSL, named OpenMLPerf, will be released in open source license allowing the use, improvement and evolution. To select of DSL support development tool, we performed a Systematic Mapping Study, where we found fifty-three (53) DSL supporting tools. We choose Ecore, from Eclipse Modeling Framework (EMF), and Sirius framework to implement the language metamodel and modeling language. An empirical evaluation was performed to verify the effort required by the user to model a performance test scenario using the OpenMLPerf approach in comparison a UML profile for performance test. The results present evidences a very close effort by the participants who model using both approachs, however, OpenMLPerf approach was affected by performance issues related to Eclipse platform.

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

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CBMG Customer Behavior Model Graphs

CE Critério de Exclusão

CI Critérios de Inclusão

CQ Critérios de Qualidade

DD Decisão de Design

DSL Domain-Specific Language

DSML Domain-Specific Modeling Language

EMF Eclipse Modeling Framework

GEF Graphical Editor Framework

GMF Graphical Modeling Framework

LW Language Workbench

MBT Model-Based Testing

MDA Model-Driven Architecture

MDE Model-Driven Engineering

MOF Meta-Object Facility specification

OMG Object Management Group

PF Pontuação Final

QoS Quality of Service

QP Questões de Pesquisa

RQ Requisito

SMS Systematic Mapping Study

SQL Structured Query Language

SUT System Under Test

TD Teste de Desempenho

UML Unified Modeling Language

UNIPAMPA Universidade Federal do Pampa

XML Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Com o número cada vez maior de aplicações web baseadas em servidores locais ou em nuvem, tornou-se imprescindível testar, prever e analisar as capacidades e o desempenho da infraestrutura que os hospeda. Assim, testes de desempenho devem ser aplicados em sistemas hospedados em máquinas físicas ou virtuais, tanto quanto em sistemas que estão hospedados na nuvem ou distribuídos de forma híbrida. A execução e a análise dos resultados dos testes aplicados em um sistema já implementado ajuda a verificar se a utilização dos recursos está sendo realizada de forma eficiente, a planejar melhorias na arquitetura e evitar desperdícios de recursos computacionais. Assim, executar testes para prever as demandas futuras de infraestrutura ajuda na elaboração de um planejamento eficiente, não só das capacidades computacionais desta infraestrutura, como das tecnologias que serão utilizadas para a implementação do sistema.

Um dos desafios para se aplicar o teste de desempenho está na integração do conhecimento do domínio de aplicação com o conhecimento das tecnologias relacionadas ao hardware e software utilizado para criação do ambiente para hospedagem do sistema. Uma das abordagens utilizadas para mitigar este desafio é a utilização da metodologia de Model-Driven Engineering (MDE) (Engenharia Dirigida por Modelos) e suas atividades, como o Model-Based Testing (MBT) (Teste Baseado em Modelos), que focam no uso de modelos para expressar o domínio e modelar o sistema responsável pela solução do problema do domínio. MDE e MBT fornecem apoio com várias atividades que teriam que ser realizadas pelo usuário final, como a geração de código e scripts. Assim os esforços do usuário ficam concentrados apenas na modelagem do domínio. Uma das atividades que apoiam esta metodologia é a utilização de linguagens para contextos específicos.

DSL (FOWLER, 2010) ou Linguagem Específica de Domínio vem sendo amplamente usada como abordagem para solucionar algum problema em um determinado domínio. É utilizada nos mais variados domínios, desde a modelagem de testes de desempenho (BERNARDINO, 2016), aprendizado de máquina (ZHAO; HUANG, 2018), design dirigido a domínio orientado a objeto (LE; DANG; NGUYEN, 2018), criação automática de tabelas de horários para escolas (RIBIĆ et al., 2018) e até linguagens para comunicação e consultas com banco de dados, como o Structured Query Language (SQL).

1.1 Motivação

Modelar testes, sem a preocupação com tecnologias usadas nas plataformas de implementação (FRANCE; RUMPE, 2007) é um dos benefícios proporcionados pelas *DSLs* para MBT.

Bernardino (2016) propôs a DSL Canopus para a modelagem de cenários e *scripts* de testes de desempenho para sistemas web. A Canopus tem como objetivo, além da modelagem dos *cenários* e *scripts* de teste, a geração automática dos códigos, para servirem como dados de entrada em ferramentas geradoras de carga, como a *HP LoadRunner* (He-

wlett Packard, 2018). A Canopus foi desenvolvida em um esforço conjunto entre a indústria e a academia, por meio de uma parceria entre os pesquisadores e uma multinacional de Tecnologia da Informação.

Para o desenvolvimento da Canopus foi utilizado uma *Language Workbench (LW)* chamada *MetaEdit+* (MetaCase, 2018). Apesar de ser considerada uma experiencia satisfatória do ponto de vista funcional, o fato de a *MetaEdit+* ser uma ferramenta distribuída com licença comercial, que requer que cada usuário da Canopus compre uma licença da *MetaEdit+*. Este fato tornou a implantação e utilização da Canopus financeiramente custosa e em muitos cenários inviável. Assim, tornou-se relevante o desenvolvimento de uma nova versão que fosse distribuída sob uma licença *open source*, tanto para uso da indústria, como para pesquisa e desenvolvimento pela academia.

1.2 Objetivo

O objetivo principal deste trabalho é desenvolver uma *DSL* gráfica sob uma licença *open source* para modelagem de cenários de testes de desempenho para sistemas web.

Para alcançar o objetivo principal, foram definidos através do levantamento de requisitos e decisões (ver Seção 5.1) os seguintes objetivos específicos:

- ⋆ Implementar uma *DSL* que contemple os seguintes requisitos gerais:
  - Licença *Open Source*
  - Possibilite a modelagem e visualização gráfica dos modelos
  - Represente as características do Teste de Desempenho (TD)
  - Modelagem de diferentes métricas e *script* para que possam ser reusadas por outros cenários
  - Modelagem de diferentes perfis de usuários e seus comportamentos
- ⋆ Avaliação através de experimento empírico

1.3 Metodologia

Para identificar quais as ferramentas existentes, as notações cobertas (gráfica ou textual) e suas licenças, foi realizado um Mapeamento Sistemático da Literatura. Foram pesquisadas as bases mais importantes para a área da ciência da computação, aplicados critérios de inclusão, exclusão e qualidade. Assim foram qualificados e classificados os estudos mais relevantes para o contexto de ferramentas de desenvolvimento de *DSL*. Com estes estudos foi possível identificar quais ferramentas são as mais adequadas para alcançar os objetivos propostos neste trabalho.

Como pode ser visto na Figura 1, o processo de desenvolvimento da *DSL* OpenML-Perf foi dividido em:

- ⋆ Concepção

  - Levantamento de requisitos da linguagem
    - Identificação dos requisitos necessários para satisfazer as necessidades do domínio de teste de desempenho
  - Decisões de design
    - Decisões necessária para atender os requisitos da DSL, *i.e.*, ferramenta para implementação, suporte a geração de código etc
  - Mapeamento Sistemático da Literatura
    - Estudo sistemático para identificação das ferramentas existentes que apoiem a criação e manutenção de DSL

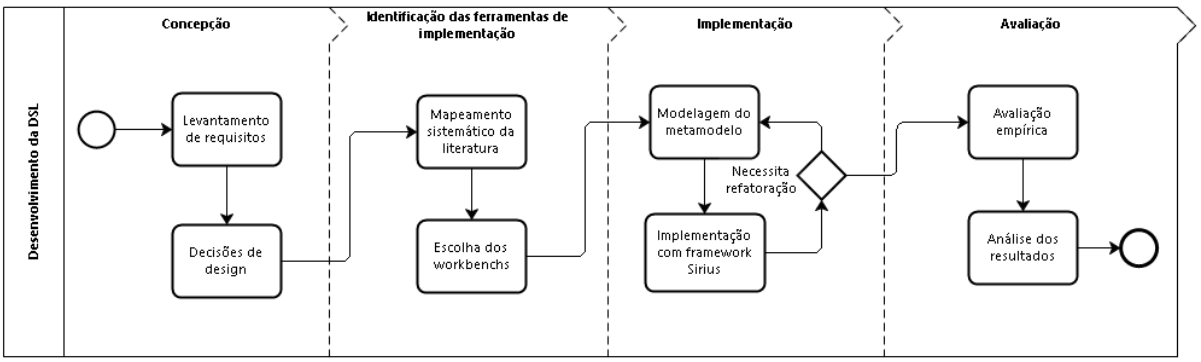
⋆ Implementação

  - Modelagem do metamodelo com *framework* Ecore
    - Definição das propriedades, relações e elementos do domínio e suas representações no metamodelo
  - Implementação da representação visual da linguagem com *framework* Sirius
    - Representação gráfica do que foi definido no modelo e que será a interface da linguagem

⋆ Avaliação

  - Avaliação empírica para avaliação do esforço, efetividade e facilidade na utilização da linguagem

Figura 1 – Processo adotado no desenvolvimento do trabalho



Fonte: Autor

1.4 Organização

Este trabalho está organizado da seguinte forma: O Capítulo 2 estabelece a fundamentação teórica para embasamento técnico e teórico sobre os assuntos e termos abordados durante o trabalho. O Capítulo 3 discute trabalhos relacionados que utilizam DSL

de alguma forma para teste de desempenho. O Capítulo 4 apresenta o Mapeamento Sistemático da Literatura e seus resultados, realizado para auxiliar a encontrar a ferramenta de apoio utilizada no desenvolvimento da linguagem proposta. O Capítulo 5 apresenta os requisitos e decisões de design, o metamodelo definido para a linguagem e a implementação da linguagem. O Capítulo 6 apresenta a avaliação empírica executado para avaliar a *DSL*. Por fim, no Capítulo 7, serão discutidos as conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

O conceito de utilizar modelos para expressar um domínio e criar software tem sido utilizado em larga escala nas últimas décadas. O uso de modelos em todas as fases do ciclo de vida do software é um dos princípios básicos da MDE. Nesse contexto se torna adequado adotar modelos para a abstração de cenários de teste e geração de código. *DSLs* permitem a modelagem de domínios e posterior geração de código, que podem ser usados como entrada para ferramentas de teste. Nesta seção serão discutidos e explicados termos que serão encontrados e utilizados nos capítulos e seções seguintes deste trabalho e que fazem do domínio do qual estão inseridas as *DSLs*.

2.1 Engenharia dirigida por Modelos (MDE)

Engenharia dirigida por Modelos (MDE), é uma abordagem da engenharia de software que utiliza modelos e suas transformações para a abstração de diversos aspectos de um domínio e/ou de um sistema. Segundo France e Rumpe (2007), um dos principais objetivos do MDE é isolar os desenvolvedores das complexidades das plataformas de implementação. Ou seja, esconder complexidades que estão ocorrendo durante a execução do sistema, como a infraestrutura, *middlewares*, bibliotecas, redes de computadores, etc. Para Kent (2002), MDE combina arquitetura, processo e análise para a abstração do domínio. Fazem parte das atividades do MDE o Model-Driven Architecture (MDA) e o MBT.

2.1.1 Teste baseado em Modelos

(Teste Baseado em Modelos) MBT é uma abordagem no qual se utiliza modelos para expressar um domínio a ser testado. Com estes modelos é possível realizar transformações e geração de código pra ferramentas de teste. Segundo Utting e Legeard (2010) existem quatro abordagens principais para MBT:

- *Geração de dados de entrada de teste para um modelo de domínio*;
- *Geração de casos de teste de um modelo de ambiente*;
- *Geração de casos de teste com oráculos de um modelo de comportamento*;
- *Geração de script de testes abstratos.*

Segundo El-Far e Whittaker (2002), as principais atividades do MBT são:

- Construir um modelo: Construir um modelo baseado nas especificações do sistema;
- Gerar as entradas esperadas: Usar o modelo para gerar entradas de teste, *i.e*, casos de testes, *scripts de teste*;

- Gerar as saídas esperadas: Gerar algum mecanismo que determina se a execução do teste está correta;
- Executar os testes: Executar os *scripts* de teste e armazenar os resultados de cada teste;
- Comparar os resultados: Comparar os resultados com as saídas que são esperadas;
- Decidir próximas ações: Ações como, estimar a qualidade do software;
- Parar o teste: Concluir o teste e liberar o software;

Para a realização destas tarefas, ferramentas de apoio devem ser adotadas, permitindo assim aproveitar todas as vantagens proporcionadas pelo MBT. Várias ferramentas para apoio ao MBT estão disponíveis, incluindo ferramentas comerciais, acadêmicas ou de licença aberta (All4Tec, 2019)(Austrian Institute of Technology, 2019)(Test Optimal, 2019).

2.1.1.1 Teste de Desempenho (TD)

Uma das atividades mais importantes da Engenharia de Desempenho de Teste (SMITH; WILLIAMS, 1993) é o *Teste de Desempenho Testing* ou TD. Que possui foco principalmente em melhorar a escalabilidade, desempenho e a descoberta de gargalos nos sistemas que estão sendo testados. O teste de desempenho é apoiado por abordagens baseadas em medições, como, análise de requisitos não funcionais e métricas de desempenho, *i.e.*, tempo de resposta, taxa de processamento. Com o teste de desempenho é possível determinar o comportamento de um sistema sob diferentes condições, identificando os limites de processamento do sistema, memória insuficiente e baixo espaço em disco. Segundo Meier et al. (2007), o teste de desempenho pode ser classificado em duas categorias:

- Teste de carga: Possui o objetivo de determinar ou avaliar o comportamento do System Under Test (SUT) sob uma carga normal de trabalho e assim determinar se o sistema está de acordo com os requisitos e especificações.
- Teste de estresse: Busca determinar o comportamento do sistema em situações onde a carga de trabalho excede o normal e assim determinar pontos de falhas e gargalos do sistema.

Ainda segundo Meier et al. (2007), o processo de executar testes de desempenho é composto de um conjunto de atividades:

- Identificar o ambiente de teste: Definir o ambiente físico para teste, *i.e.*, hardware, software e configurações de rede.

- Identificar os critérios de aceitação de desempenho: Identificar objetivos e suposições em relação ao tempo de resposta e uso de recursos computacionais do sistema.

- Planejar e modelar os testes: Identificar cenários e a variabilidade das representações de usuários.

- Configurar o ambiente: Preparar o ambiente de teste, ferramentas e recursos necessários para realizar o teste de desempenho.

- Implementar a modelagem do teste: Desenvolver os *scripts* e cenários de teste.

- Executar os testes: Implementar o monitoramento de desempenho e rodar os testes.

- Analisar os resultados e refazer os testes: Resumir, consolidar e compartilhar os resultados dos testes.

2.1.2 Linguagens Específicas de Domínio

Linguagens Específicas de Domínio, ou Domain-Specific Languages *DSL*, são também conhecidas como *Little Languages, Small Languages*, Linguagens de Propósito Especial ou *Domain-Specific Modeling Language (DSML)*. Como Mernik, Heering e Sloane (2005) define, *DSLs* "são linguagens de programação para computadores com expressividade limitada que possuem foco em domínio especifico" Mernik, Heering e Sloane (2005)(tradução). Enquanto isso, *DSML* são linguagens específicas de domínio com sintaxe diferente, que usa componentes gráficos para modelar o comportamento do sistema Kelly e Tolvanen (2007). Uma *DSL* pode ser classificada de duas formas: *DSL* interna ou *DSL* Externa.

2.1.2.1 DSL Interna:

Uma *DSL* pode ser classificada como interna quando usa a estrutura de uma linguagem hospedeira. Como foi definido por Fowler (2010), "uma *DSL* interna é uma maneira Específica de usar uma linguagem de propósito geral... um script, em uma *DSL* interna, é um código válido em sua linguagem de propósito geral." Fowler (2010) (traduzido).

2.1.2.2 DSL Externa:

Uma *DSL* externa é uma linguagem independente da linguagem hospedeira da aplicação. Em uma *DSL* externa um *script* pode ser analisado por um código na aplicação, usando técnicas como analises sintática e semânticas (*e.g* Structured Query Language - SQL) Fowler (2010).

2.1.2.3 *Language Workbench* (LW)

São ferramentas utilizadas para desenvolver DSLs. De acordo com Wachsmuth, Konat e Visser (2014a), LWs fornecem "mecanismos de alto nível para a implementação de linguagens de programação tornar acessível o desenvolvimento de novas linguagens" Wachsmuth, Konat e Visser (2014a) (traduzido). São ambientes que além de usados para criação de novas *DSLs*, trabalham em conjunto com outras ferramentas necessárias para o uso eficiente destas DSLs por parte dos usuários finais. LWs facilitam não apenas a definição de analisadores sintáticos, mas também ajudam a criar ambientes customizados para as linguagens. Além disso, permitem a definição de sintaxes abstratas, as quais são comumente usadas para obter a linguagem desejada, que pode ser acessível em um ambiente integrado de desenvolvimento Korenkov, Loginov e Lazdin (2015a).

2.1.2.3.1 Framework

São conjuntos conceitos usados em um domínio específico para solucionar problemas. De acordo com Johnson (1997), "um *framework* é um esqueleto de uma aplicação que está apta a ser customizada por um desenvolvedor" Johnson (1997). *Frameworks* são "componentes" e uma aplicação pode usar vários *frameworks*. Entretanto, *frameworks* podem ser adaptados de acordo com as necessidades do usuário, fornecendo mais flexibilidade que outros componentes Johnson (1997).

2.1.2.4 Modelos, Metamodelos e Meta-Metamodelos

Segundo Fuentes-Fernández e Vallecillo-Moreno (2004), o conceito de organização de modelos, metamodelos e meta-metamodelos baseado na *Meta-Object Facility specification (MOF)* da Object Management Group (OMG) (ver Figura 2) é classificado em quatro camadas.

Camada 0 (M0): Esta camada é nomeada como "instância", sua função é modelar o sistema em execução. Seus elementos são instâncias reais que existem no sistema. Exemplos destas instâncias são: "Shneider" que vive no "Alabama", comprou uma cópia do livro "DSL".

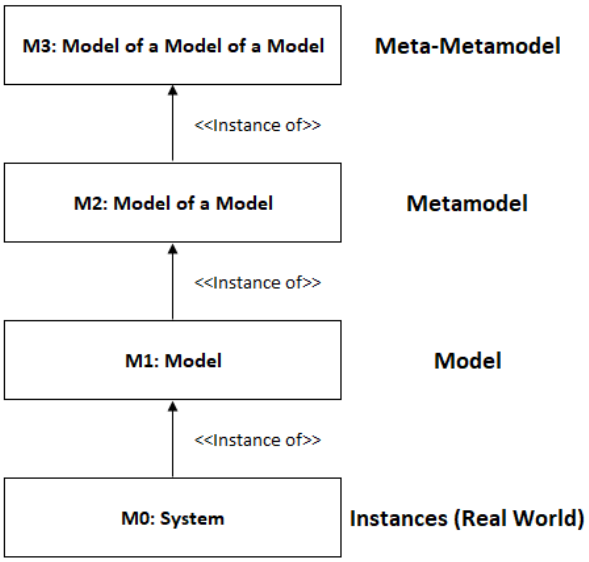
Camada 1 (M1): Modelo é uma descrição de um sistema (ou parte dele) escrita em uma linguagem bem definida. Uma linguagem bem definida é uma linguagem com sintaxe e semântica adequadas para a interpretação automática por um computador. Exemplos desta instância são: "Pessoa", "Estado" e "Livro".

Camada 2 (M2): Modelo de um modelo (Metamodel) são elementos que compreendem a linguagem de modelagem. Esta camada define o conceito que será usado para modelar um elemento da comada de modelo e cada elemento da camada de modelo

é uma instância de um elemento no metamodel. Considerando a UML, por exemplo, "Classe", "Atributos" e "Relações" são definidos na M2.

Camada 3 (M3): Modelo de um modelo de um modelo (Meta-Metamodelo) define o conceito que pode ser usado para definir linguagens de modelagem. *e.g.* UML define "Classe" como um classificador.

Figura 2 – Arquitetura de meta modelagem baseada no *MOF*da OMG.



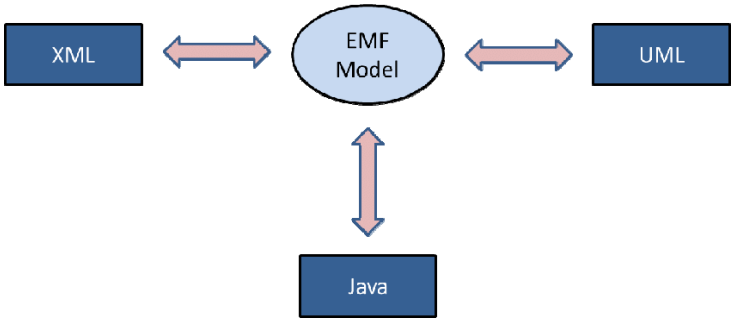
Fonte: Tang (2009)

Frameworks e LWs podem ser classificados como metamodelos respeitando as Especificações contidas no *MOF* do *OMG*e *DSL*, entretanto, são classificadas como modelos.

2.1.3 Eclipse Modeling Framework (EMF)

Desenvolvido e mantido pela Eclipse Foundation, o *EMF* é um *framework* que possibilita a abstração e modelagem de domínios através da definição de modelos. As modelagens podem ser realizadas através de ferramentas de modelagem da Unified Modeling Language (UML), como diagramas de classe, ou anotações JAVA e através de esquemas de Extensible Markup Language (XML), permitindo a geração de código a partir dos modelos definidos. Segundo Steinberg et al. (2008), o *EMF* permite a unificação de três tecnologias importantes (ver Figura 3), o Java, XML e UML, pois independentemente de qual das tecnologias foi usada para definir o modelo, o EMF agrupa todas em torno de um modelo *EMF* que as mantém unidas. O Eclipse EMF é distribuído de forma aberta sob a licença Eclipse Public License 2.0

Figura 3 – EMF unifica UML, JAVA e XML



Fonte: Steinberg et al. (2008)

2.1.3.1 Ecore

Ecore é o modelo usado para representar outros modelos no EMF. Segundo Budinsky et al. (2004), Ecore é seu próprio metamodelo, uma vez que o Ecore é um modelo EMF.

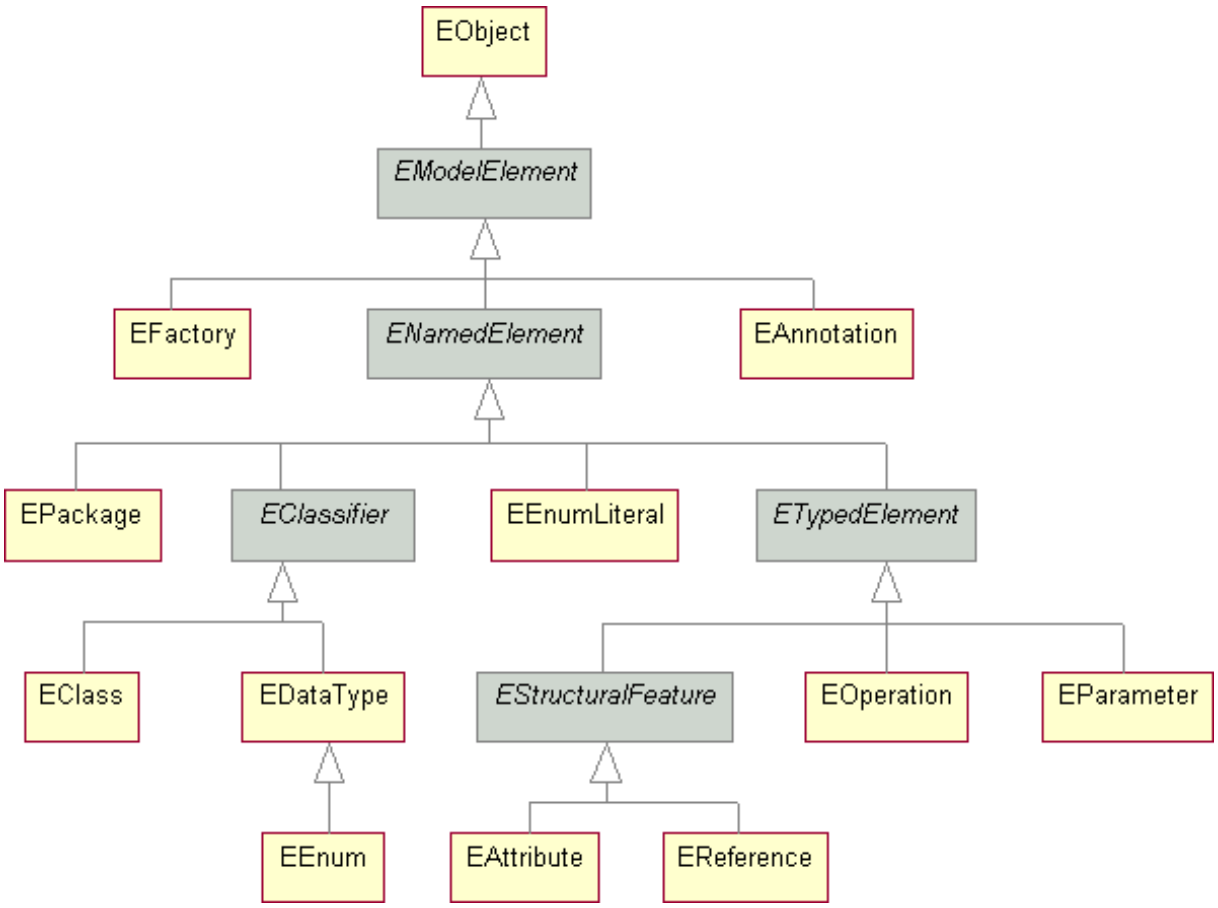
O conceito dos componentes do modelo Ecore são arranjados seguindo uma hierarquia baseada em um dialeto da UML. O componente superior é chamado **EObject**, dele deriva outro componente abstrato chamado **EModelElement**. Neste ponto da árvore hierárquica do Ecore encontra-se o componente **ENamedElement**, que do quais todos os componentes abaixo na hierarquia, com exceção de **EAnnotation** e **EFactory** que estão ao lado, herdam a definição do atributo ***name***. Abaixo deste nível vem outros componentes como **EClassifier**, **ETypedElement**, **EClass**, **EDataType**, **EAtribute**, **EReference**, entre outros (ver Figura 4).

2.1.4 Eclipse Sirius Framework

O *framework* Sirius é um projeto de licença aberta pertencente e mantido pela Eclipse Foundation, cujo objetivo é permitir a criação de ferramentas de modelagem. Sirius encapsula o EMF, Graphical Editor Framework (GEF) e Graphical Modeling Framework (GMF). O *EMF* fornece um objeto gráfico em forma de modelo para representação, o GEF provê a capacidade de construção de representações visuais dos modelos e o *GMF* fornece a ferramenta para a criação de editores gráficos Viyović, Maksimović e Perišić (2014). No Sirius, os editores são definidos por um modelo que tem objetivo de estabelecer o comportamento de todas as ferramentas de edição e navegação utilizadas para adaptar uma *DSL*.

A construção da ferramenta de modelagem, ou *DSL*, é realizada seguindo as definições da modelagem do domínio do problema estabelecidas em um modelo ECore. É possível estabelecer representações gráficas, tanto para relações, quanto para elementos (ver Figura 5). Sendo possível escolher partes do modelo que irão ser representadas e modeladas e partes que tem um significado semântico no modelo, mas ficaram escondidas

Figura 4 – Estrutura hierárquica do ECore



Fonte: Eclipse Foundation (2018a)

do usuário, apenas servindo como regras do modelo.

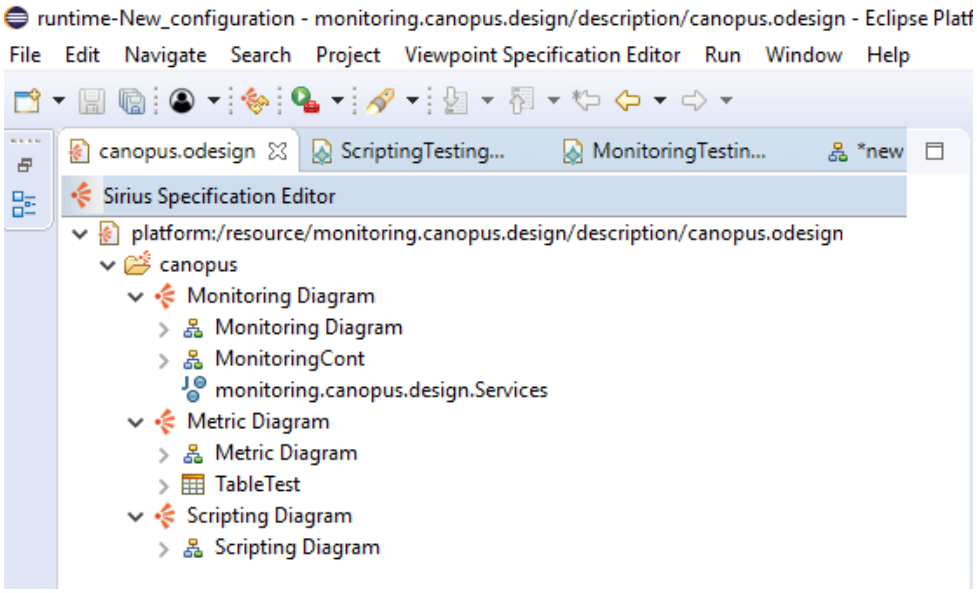
2.1.5 DSL Canopus

A *DSL* Canopus (BERNARDINO, 2016), com foco na modelagem de testes de desempenho para sistemas web, foi desenvolvida dentro do *LW* MetaEdit+ (MetaCase, 2018), ferramenta criada e mantida pela MetaCase. A Canopus permite a modelagem de diversos aspectos do domínio de teste de desempenho. Como a definição de perfis de usuários, escalabilidade de acessos e recursos, monitoramento de *SUTs* e suas métricas associadas. Também permite a modelagem dos *scripts* que descrevem as atividades e comportamento dos perfis de usuários modelados. Estas informações são associadas à modelagem dos cenários em que estes perfis e *script* serão executados.

2.1.6 Lições do capítulo

Neste capítulo, foram explorados os principais conceitos que estão envolvidos em nosso trabalho. O MDE, MBT e principalmente DSL foram explorados, buscando compre-

Figura 5 – Interface gráfica do *framework* Sirius



Fonte: Autor

ender o impacto e onde se encaixa nossa proposta dentro do processo de desenvolvimento de software. A utilização de modelos para representação de conceitos e propriedades vem sendo amplamente utilizada pela industria e academia. Assim, deve ser cada vez mais observado o uso destas representações do que a utilização das tecnologias executoras em si. O MBT, se insere neste contexto de forma natural, pois todo sistema, mesmo que ainda não implementado, deve ser testado e a melhor forma de testar um sistema ainda não existente é através de modelos que representem aspectos do domínio. Assim, discutir conceitos, como, modelo, metamodelo e meta metamodelo, auxiliam para uma maior compreensão dos mecanismos de implementação da linguagem e de sua integração com o mundo real. Compreender estes conceitos auxilia não apenas na implementação da nossa proposta, como também na escolha da ferramenta mais adequada para a realização desta implementação.

No Capítulo seguinte, serão apresentados os trabalhados relacionados a nossa proposta. Serão discutidos trabalhos que utilizem DSL em alguma etapa do processo de teste de desempenho. Também serão apresentadas considerações sobre ferramentas execução de teste de desempenho.

3 TRABALHOS RELACIONADOS

É possível encontrar muitos trabalhos relacionados a DSLs nos mais variados domínios (NAKAMURA et al., 2012), (PEREZ; VALDERAS; FONS, 2013), (RIBIĆ et al., 2018). Sabe-se que, até a data da execução deste trabalho, apenas a DSL Canopus (BERNARDINO, 2016), foi desenvolvida para realizar modelagem com notação gráfica de testes de desempenho. Neste capítulo serão apresentados e discutidos algumas ferramentas para teste de desempenho, sendo que, algumas são DSLs propriamente ditas e outras utilizam DSLs para auxiliar a ferramenta em seu propósito.

3.1 Trabalhos relacionados

Ruffo et al. (2004) em seu trabalho, apresenta uma ferramenta para avaliação de desempenho de aplicações web com foco no comportamento do usuário. Chamada de *WALTy (Web Application Load based Testing tool)*, é formada por um conjunto de ferramentas que permite a análise escalável dos testes. Para isso, os testes de carga são baseados em informações retiradas dos *logs* das aplicações através do uso de *Customer Behavior Model Graphs (CBMG)*, que são propostas para a caracterização de grafos de transição de estado de sites de *e-Commerce*. CBMG são usados para representar cada seção, para depois um algoritmo *clusterizado* é usado para reunir todas as seções identificadas. Assim são identificados os perfis de usuário de um grupo de usuários que possuem padrões de navegação em comum. O processo da *WALTy* pode ser estruturado da seguinte forma: Primeiramente, construtores CBMG, através dos dados de entrada (*logs*), realizam a definição de estados em um conjunto de regras de um usuário, definem objetos embarcados e onde esses objetos devem ser filtrados e geram especificações dos parâmetros, para assim, caracterizar o comportamento do site do usuário, através do número de *cluster's* e limite do tempo de sessão; Após isto, é gerado a representação do trafego da rede, isto é feito através de um CBMG com o uso de um algoritmo. Um dos fatores que mais diferenciam este trabalho do proposto por nós, é o fato do momento de realização do teste. Enquanto no trabalho de Ruffo et al. (2004) o sistema precisa estar totalmente implementado, no nosso, é necessário apenas a ideia de sistema, uma vez que o que é modelado é o domínio da aplicação e o conceito dela, assim ela pode ser pré-modelada e testada antes de sua implementação. Além disto, nossa abordagem é uma abordagem visual, o que torna ela muito mais universal em relação as tecnologias a serem adotadas para a execução do teste.

No trabalho de Abbors et al. (2012), a ferramenta baseada em modelo para teste de desempenho MBPeT é apresentada. A ferramenta usa modelos para gerar cargas de trabalho a serem aplicadas em tempo real nos sistemas, assim, coletando diferentes medidas de desempenho. Os modelos são definidos usando autômatos probabilísticos cronometrados, assim descrevendo diferentes perfis de usuários que interajam com o sistema. Como entrada, a ferramenta aceita um conjunto de modelos expressos como autômatos, o

número de usuários virtuais, as funções de *ramp* e duração do teste. Em seguida fornece um relatório descrevendo as medidas do teste. Os modelos de desempenho são compostos por: Comportamento do usuário, descrito como, localizações e transações. Os modelos são lidos por *parsers*, para então, construir as representações do modelo e validá-los através de um conjunto de regras, como as conexões das localizações serem válidas. O teste é relatado por um módulo criador de *report*, que cria um *HTML* com todos os resultados da execução do teste. O gerador de carga envia uma carga para o SUT, cada instância de um modelo contém ações em sequência do usuário, expresso em probabilidades. Assim como no trabalho de Ruffo et al. (2004) citado anteriormente, também é necessário o sistema já estar definido e construído e não há representação gráfica para todas as etapas do teste, apenas para os modelos dos autômatos. Enquanto que em nossa abordagem, o aspecto visual contempla todos as etapas da modelagem do teste de desempenho.

Uma abordagem declarativa para execução de testes de desempenho em ambientes de desenvolvimento de software continuo, é apresentada por Ferme e Pautasso (2018). A abordagem consiste em uma DSL para teste de desempenho e um framework programável por ela, dirigida ao processo fim-a-fim de execução destes testes. Os testes podem ser especificados a partir de modelos (*templates*) de testes padrões ou modelos mais avançados. A DSL foi implementada como uma expressiva linguagem orientada a objetivo. Os testes são especificados de forma declarativa, com execuções automatizadas e orientadas por modelos. A segunda parte da abordagem, relacionada ao framework, utiliza a própria DSL para programar e automatizar o processo de execução do teste, executando todas as atividades e objetivos que o usuários especificou para a execução do teste. A DSL proposta na abordagem é utilizada para especificar de forma declarativa os objetivos do teste e controlar os processos de execução, sendo adaptada para sistemas de micro serviço. Algumas das características de desempenho que podem ser especificadas com a DSL são: Funções de carga, cargas de trabalho, usuários virtuais, dados para teste, gerenciamento de conjuntos de teste e análise de dados de desempenho. Segundo os autores, o framework foi construído para auxiliar os usuários em todas as atividades do teste de desempenho. A abordagem proposta pelos autores apresenta uma DSL textual, com linguagem declarativa, muito presa a termos do teste de desempenho, o que pode torná-la não muito amigável a usuários do domínio da proposta. Nossa proposta, também utiliza termos e propriedades do teste de desempenho, mas consideramos muito mais acessível à usuários do domínio da proposta, o de aplicações web. Além disso, por ser visual, nossa proposta complementa a informação de atributos e propriedades, com representações gráficas do domínio e relações entre elementos.

Sun, White e Eade (2014) apresentam uma DSL textual para especificar planos de teste de carga em alto nível e requisitos para os chamados *Quality of Service (QoS)*, sem detalhes das configurações de baixo nível. A DSL chamada *GROWL*, foi implementada com a utilização do framework Xtext. A partir da DSL, é construído um modelo que gera

especificações de teste compatíveis com a ferramenta de teste de carga jMeter. A DSL é utilizada para a captura da configuração de recursos e informações sobre os QoSs, não capturadas pelas ferramentas de geração de carga. São usadas analises de modelo para identificar a relação entre o desempenho de QoSs e as configurações de recursos e a geração de código para a alocação automática destas configurações. Através de uma instância de um modelo *GROWL*, uma especificação baseada em XML é gerada, como o auxílio do xTend. A *GROWL* satisfaz a parte de modelagem, análise do modelo e geração de código, dentro da abordagem ROAR (Resource Optimization, Allocation and Recommendation System), proposta pelo trabalho, que combina a modelagem, análise do modelo, automação do teste, geração de código e técnicas de otimização. Em relação ao proposto em nosso trabalho, o trabalho de Sun, White e Eade (2014) não permite a modelagem de comportamentos complexos através de parâmetros de entrada, *i.e.*, parâmetros variáveis como entrada dos testes. Uma vez que, os parâmetros são especificados na modelagem, enquanto na nossa abordagem, os parâmetros podem ser dinâmicos, oriundos de arquivos externos.

O trabalho de Spafford e Vetter (2012) descreve uma abordagem para modelagem de desempenho analítico que utiliza uma DSL chamada Aspen. Aspen especifica uma gramática formal para descrever dois tipos de modelos, um descreve o comportamento de uma aplicação, incluindo paralelismo, contadores, estruturas de dados e fluxos de controle, o outro especifica um modelo de máquina abstrata. Ambos modelos especificados são compilados, checados para corretude semântica e persistido ou simulados para cenários de modelagem. Ao contrário da OpenMLPerf, que possui foco em modelos de medição que podem ser comparados com outras técnicas, Aspen considera modelos preditivos para comparar o desempenho com outras técnicas, como modelos analíticos.

Algumas ferramentas de apoio à testes de desempenho têm sido desenvolvidas. A Gatling (2018) é uma ferramenta de teste de carga, em que os cenários são definidos em código, baseados em uma *DSL* expressiva, em que os cenários são autoexplicativos. A Load Runner da Hewlett Packard (2018) realiza medições do comportamento de um sistema sob carga. Simulando as atividades do usuário ao gerar mensagens entre as aplicações. Ambas as soluções permitem a simulação de cenários e geração dos testes de desempenho. Mas ao contrário da proposta deste trabalho, nenhuma permite a modelagem dos cenários de teste de forma gráfica, independente das tecnologias de teste de carga e desempenho.

3.1.1 Lições do capítulo

Utilizar DSLs para representar testes de desempenho tem ganho cada vez mais espaço na hora de decidir qual abordagem utilizar pra representar os modelos do domínio de teste. Tanto ferramentas compostas por um subconjunto de outras ferramentas, incluindo DSLs, como a utilização unicamente de linguagens especificas. Mas apesar desta variedade de ferramentas de modelagem, não foi observada nenhuma que permitisse, como no

caso da OpenMLPerf e da DSL canopus, a modelagem gráfica de testes de desempenho, existindo em sua grande maioria, ferramentas que permitem modelagem através do uso da notação textual.

Neste capítulo foram apresentados trabalhos relacionados à nossa proposta de ferramenta de modelagem de teste de desempenho em sistemas web. Alguns trabalhos fazem uso de DSLs em algum momento do processo de desenvolvimento, execução ou uso do usuário da linguagem. Também foram apresentadas algumas ferramentas para execução de teste de desempenho, que podem vir a ser futuras ferramentas de saída para a OpenMLCanopus.

4 MAPEAMENTO SISTEMÁTICO DA LITERATURA

Foi realizado um mapeamento sistemático da literatura com o objetivo de identificar ferramentas de desenvolvimento de DSLs (LWs) de licença aberta e que fornecessem suporte à criação de *DSL* com notação gráficas. Neste capítulo serão discutidos o protocolo do SMS, seu planejamento e decisões, assim como a execução e os resultados obtidos.

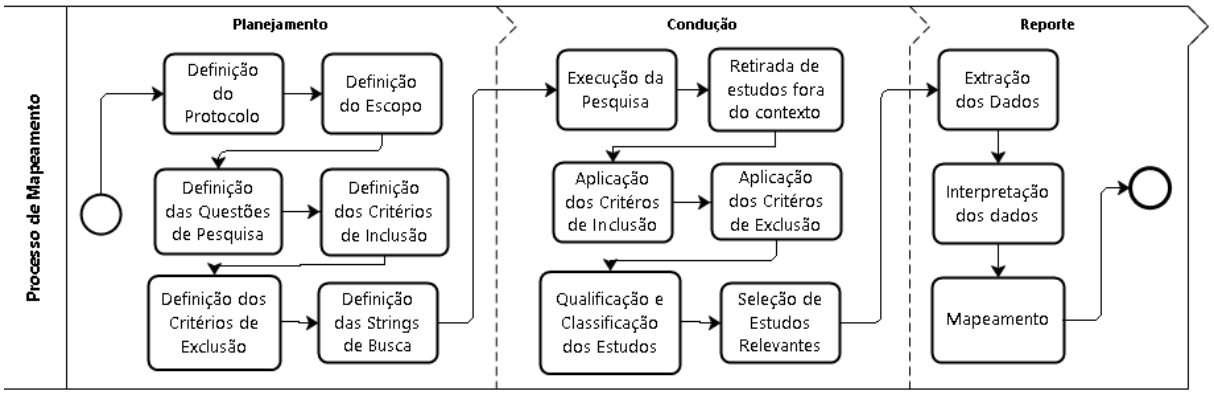
4.1 Processo do SMS

Neste SMS, foi seguido um estruturado e bem estabelecido processo de revisão, apresentado por Petersen et al. (2008), para evitar retrabalho e perda de informação relevante. Este processo pretende definir todos os aspectos necessários para a execução precisa do SMS e obter os resultados pretendidos.

4.1.1 Definição do protocolo

Na Figura 6, são apresentadas as macro frases do processo: `Planejamento`, `Condução` e `Resultados`. A primeira fase (`Planejamento`) e suas tarefas são detalhadas nesta seção. A segunda fase, `Condução` e as atividades de execução, são apresentadas na Seção 4.2. Subsequentemente, a Seção 4.3 apresenta e discute os resultados obtidos na atividade de extração dos dados do SMS.

Figura 6 – Processo de Mapeamento Sistemático da Literatura criado com notação BPMN



Fonte: Autor: Baseado em Petersen et al. (2008)

4.1.2 Definição do Escopo

Nosso objetivo de pesquisa é mapear ferramentas de suporte, LW e *frameworks* que apoiem a criação de *DSL* gráficas e que tenham sido citadas na literatura entre os anos de 2012 e 2017.

Este SMS busca identificar as notações que estas ferramentas apoiam no desenvolvimento de *DSL* e suas licenças. Para assim identificar a ferramenta mais adequada para

o desenvolvimento de uma *DSL* gráfica de licença *open source* . Sendo interesse deste estudo, as ferramentas usadas ou desenvolvidas pela indústria, bem como as ferramentas de pesquisa desenvolvidas pela academia. O tipo de licença na qual a ferramenta foi lançada será identificada como comercial ou *open source* .

Definição das Questões de Pesquisa

Com este objetivo, foram formuladas as seguintes Questões de Pesquisa (QPs):

QP1. *Quais são as ferramentas/frameworks/Language Workbench usadas para o desenvolvimento de DSL?*

QP2. *Quais são as notações (gráfica e/ou textual) que estas ferramentas dão suporte?*

QP3. *Sob quais licenças estas ferramentas foram lançadas?*

4.1.3 Definição dos Critérios de Inclusão e Exclusão

A atividade de definição dos critérios de inclusão e exclusão foi baseada no escopo, objetivos do estudo e nas questões de pesquisa. A pesquisa foi direcionada para os estudos mais relevantes com o contexto deste SMS, excluindo artigos sem relevância. Este passo é importante, pois determina quais trabalhos podem ser classificados e qualificados em estágios posteriores do estudo. A inclusão de estudos primários precisa atender todos os Critérios de Inclusão (CI) desta pesquisa para posterior análise e classificação. Similarmente, se um estudo é classificado em ao menos um Critério de Exclusão (CE), ele deve ser removido dos estudos primários.

Definição dos Critérios de Inclusão (CI)

CI1. *O estudo primário deve apresentar uma abordagem, técnica, método, processo, ferramenta, framework , ou LW para manipular DSL ou DSML.*

CI2. *O estudo primário deve mencionar uma ferramenta ou framework de apoio a DSL, ou LW.*

CI3. *O estudo primário deve focar na apresentação de uma DSL e mencionar a ferramenta de apoio a DSL.*

Definição dos Critérios de Exclusão (CE)

CE1. *Estudos publicados antes de 2012.*

CE2. *Estudos em qualquer idioma que não seja o inglês.*

CE3. *Duplicados e/ou incompletos.*

CE4. *Estudos apenas disponíveis na forma de resumos, apresentação de slides, posters e short paper.*

4.1.4 Processo de Busca

Durante o processo de busca foi apenas considerada o uso de base de dados da ciência da computação que fornecessem motores de busca bem estabelecidos que permitissem o uso de palavras-chaves. As seguintes bibliotecas digitais na área da computação foram usadas: Compendex (Engineering Village)[1], IEEE Xplore[2], ScienceDirect[3], Association for Computing Machinery (ACM) Digital Library[4], SCOPUS[5] and SpringerLink[6].

4.1.4.1 Definição das strings de busca

Para definir a string usada na busca, foram usados termos e sinônimos apresentados na Tabela 1. Como mostrado na 7, foi usado operadores "OR" para incluir sinônimos na string. Também foram usados operadores "AND" para adicionar mais termos. Em algumas base de dados, foram adotadas estratégias para criar diferentes versões da string, o qual é similar a string apresentada na Figura 7 (ver Seção 4.2). Para avaliar a qualidade e abrangência da string de busca, um estudo exploratório foi realizado, no qual três estudos de controle foram definidos, os quais são: Erdweg et al. (2013b), Erdweg et al. (2015a), Bernardino et al. (2017).

Tabela 1 – Definição das strings de busca

Termo	**Sinônimo**
Domain-Specific Language	DSL, DSML, Domain Specific Language, Domain-Specific-Language, Domain Specific Modeling Language, Domain-Specific Modeling Language, Domain-Specific-Modeling-Language, Small Language, Small-Language, Little Language, Little-Language
Tool	*Language Workbench*

4.1.5 Avaliação da Qualidade dos Estudos

Um conjunto de Critérios de Qualidade (CQ) foi definido, relacionados a qualidade dos estudos primários remanescentes após a aplicação dos critérios de inclusão e exclusão. estes critérios buscam quantificar a relevância de cada estudo primário e permitir a comparação dos estudos primários selecionados. Estudos primários com zero (0) pontos serão removidos da relação dos estudos primários, mesmo que sejam do domínio da pesquisa.

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

Figura 7 – String de Busca

```
(DSL OR DSML OR Domain Specific Language OR
Domain-Specific Language OR Domain-Specific-Language
OR Domain-Specific Modeling Language OR Domain
Specific Modeling Language OR
Domain-Specific-Modeling-Language OR Small Language OR
Small-Language OR Little Language OR Little-Language
AND Tool OR Language Workbench)
```

Além disso, estudos primários que não pontuaram com no CQ1, consequentemente não marcam pontos nos outros critérios, sendo então removidos da lista de estudos primários. Foi aplicado o CQ de acordo com o seguinte esquema: **S**im: 1.0; **P**arcialmente: 0.5; **N**ão: 0.0. O CQ ficou definido assim:

CQ1. *Este estudo apresentou uma ferramenta que apoie o desenvolvimento de DSL?*
Avaliação: **S**: O estudo apresenta uma ferramenta que apoie o desenvolvimento de *DSL*; **P**: O estudo menciona uma ferramenta que apoie o desenvolvimento de *DSL*, mas não apresenta detalhes sobre a ferramenta; **N**: O estudo não apresenta ou menciona uma ferramenta que apoie o desenvolvimento de *DSL*;

CQ2. *A ferramenta apoia ao menos uma das notações (gráfica ou textual)?*
Avaliação: **S**: O estudo apresenta uma ferramenta que apoie as notações gráfica ou textual; **P**: O estudo não apresenta uma ferramenta de apoio, mas os resultados indicam que a notação gráfica ou textual foram usadas; **N**: O estudo não indica evidências no uso das notações gráfica ou textual.

CQ3. *O estudo demostra como a ferramenta foi aplicada no desenvolvimento de DSL?*
Avaliação: **S**: O estudo apresenta a ferramenta usando exemplos e trechos códigos ou imagens; **P**: O estudo discute o uso da ferramenta, mas não apresenta detalhes da implementação; **N**: O estudo não apresenta qualquer evidência de implementação.

CQ4. *O uso da ferramenta é descrito de forma clara/detalhada?*
Avaliação: **S**: O estudo apresenta um tutorial, um processo, passos para completar uma tarefa, ou define um fluxo para alcançar um objetivo; **P**: O estudo apenas menciona atividades que precisam ser executadas, mas sem maiores explicações; **N**: O estudo não apresenta qualquer evidência em relação a suporte ao uso da ferramenta;

4.1.6 Estratégia de extração de dados

Foi criado um formulário[7] para identificar e extrair dados relevantes dos estudos selecionados. Esta informação tem sido usada para responder as QPs. Foi extraído de

[7] <https://tinyurl.com/SMS-DSL>

cada estudo os seguintes dados;

- ⋆ Base de dados: ACM, Compendex (Engineering Village), IEEE, SCOPUS, ScienceDirect and SpringerLink
- ⋆ Fonte: referência completa em conferencia, livro, nome do periódico
- ⋆ Título
- ⋆ Resumo
- ⋆ Autores
- ⋆ Ano
- ⋆ Ferramenta, *Frameworks*, *Language Workbench*
  - Funcionalidade: notação
  - Data de lançamento
  - Tipo de licença: commercial ou open source

4.1.7 Análise dos dados

Foram analisados os dados coletados para: Identificar as ferramantas/ *frameworks*/ *Language Workbench*es que apoiem o processo de desenvolvimento de DSL **(QP1)**; identificar as licenças destas ferramentas (comercial ou *open source*) e demonstrar o tipo de notação em relação ao seu tipo de licença **(QP2), (QP3)**.

4.2 Condução do SMS

A condução do SMS foi realizada em fevereiro de 2018. Antes deste período, o protocolo foi definido. seguido pelo mapeamento dos dados e conclusão do estudo. A pesquisa foi realizada nas bases mencionadas (ver Seção 4.1.4), respeitando as estratégias adotadas por cada motor de busca.

Como pode ser visto na Tabela 2, duas strings de busca foram usadas na base da IEEE Xplore. Foi adotada esta estratégia devido a limitação da quantidade de termos em quinze (15) por string de busca. Assim, foi dividida a string de busca em duas composições, no qual *DSL* e *DSML* são os termos principais para cada uma. mantendo os termos comuns em ambas as composições.

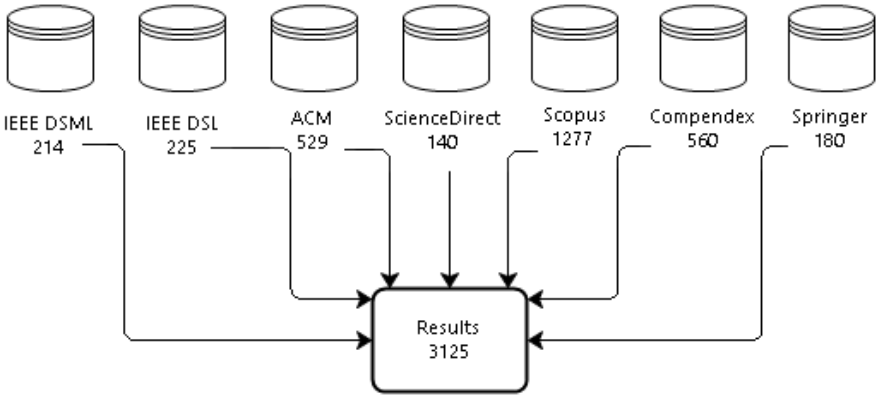
Para todas as bases de dados, a busca foi limitada aos campos “Abstract”, “Title” e “Keywords”, limitando por ano diretamente no motor de busca, neste caso, de 2012 à 2017 (incluindo). a string de busca para cada base é apresentada na Tabela 2.

Foram obtidos das bases de dados um total de três mil cento e vinte e cinco (3125) trabalhos. Oriundos da biblioteca digital da IEEE Xplore, foram recuperados duzentos e quatorze (214) trabalhos considerando o termo *DSL* como principal e duzentos e vinte e cinco (225) considerando o termo DSLs. Na biblioteca digital da ACM, um total de quinhentos e vinte e nove (529) trabalhos foram recuperados. A Science Direct recuperou cento e quarenta (140) trabalhos, enquanto a Scopus retornou mil duzentos e setenta e

sete (1277), sendo a base de dados com o maior número de retornos. Na base de dados da Compendex, quinhentos e sessenta (560) trabalhos foram recuperados e cento e oitenta (180) trabalhos retornaram da base de dados da Springer, completando um total de três mil cento e vinte e cinco (3125) trabalhos.

Após a remoção dos duplicados, como ilustrado na Figura 9, restaram mil quatrocentos e três estudos (1403) trabalhos. Assim, trabalhos considerados fora do contexto foram removidos, sendo estes artigos que continham termos como *DSL* e *DSML* mas não faziam parte do escopo da pesquisa. Neste passo, setenta e um (71) foram removidos, restando um total de mil trezentos e trinta e dois (1332) trabalhos para a aplicação dos CIs e CEs.

Figura 8 – Estudos primários resultantes das buscas nas bases de dados



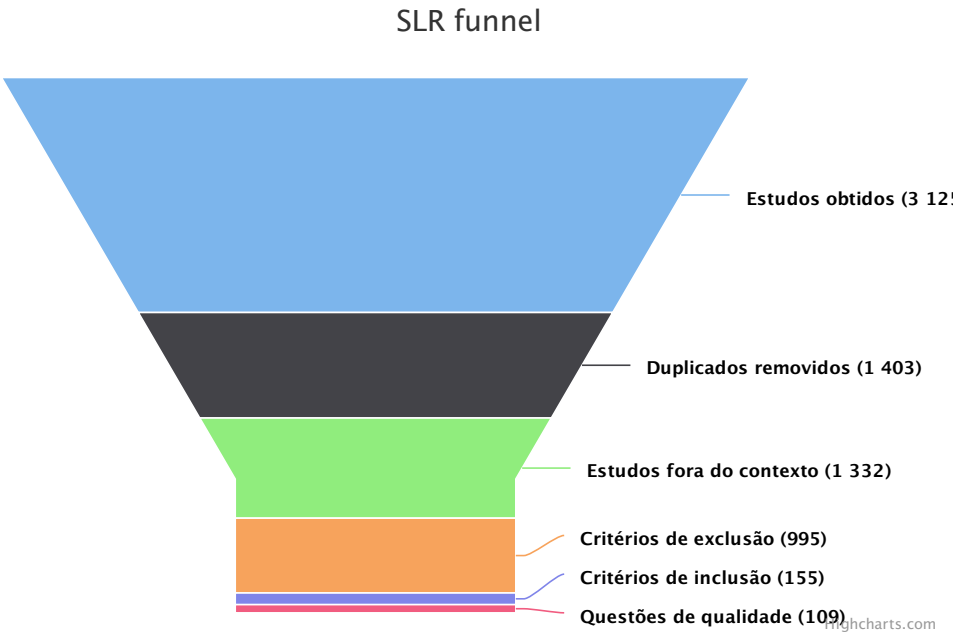
Fonte: Autor

Durante esta atividade, mil trezentos e trinta e dois (1332) trabalhos foram lidos. E como resultado, trezentos e trinta e sete (337) trabalhos foram excluídos após serem aplicados os CEs, restando novecentos e noventa e cinco trabalhos (995). Após isso, oitocentos e quarenta (840) trabalhos foram excluídos porque não contemplavam nenhum CI. Ao final da aplicação dos CIs e CEs, restaram o total de cento e cinquenta e seis (156) trabalhos para a etapa de classificação de qualidade.

Com o objetivo de comprovar a qualidade e relevância dos trabalhos associados a esta revisão, foram aplicados os CQ que são apresentados na Seção 4.1.5 .

Os resultados da aplicação dos CQs são apresentados na Tabela 3, onde cada trabalho foi identificado pela coluna (ID). As "referências"dos estudos são apresentadas na coluna nome da coluna e a coluna nome da coluna apresenta o ano em que os trabalhos foram publicados. A pontuação com relação aos critérios de qualidade de cada trabalho avaliado pelos pesquisadores é apresentada nas colunas 1,2,3 e 4. A pontuação final pode ser visualizada na coluna Pontuação Final (PF). Dois pesquisadores avaliaram individualmente cada um dos (156) trabalhos de acordo com as quatro questões de qualidade.

Figura 9 – Resultado da condução do SMS



Fonte: Autor

Tabela 2 – Strings de Busca por Base de Dados

Base de Dados	**Strings de Busca**
ACM Digital Library	((acmdlTitle:("DSLDSMLDomain Specific Language Domain-Specific LanguageDomain-Specific-LanguageDomain Specific-LanguageDomain-Specific Modeling Language Domain Specific Modeling LanguageLittle-Language Little LanguageSmall-LanguageSmall Language"))) OR (recordAbstract:("DSLDSMLDomain Specific Language Domain-Specific LanguageDomain-Specific-LanguageDomain Specific-LanguageDomain-Specific Modeling LanguageDomain Specific Modeling LanguageLittle-LanguageLittle Language Small-LanguageSmall Language")) AND ((acmdlTitle:("Tool ToolsLanguage Workbench")) OR (recordAbstract:("Tool Language Workbench")))
Compendex (Engine-ering Village)	(((DSL) OR (DSML) OR ("Domain Specific Language") OR ("Domain-Specific Language") OR ("Domain-Specific-Language") OR ("Small Language") OR ("Small-Language") OR ("Little-Language") OR ("Little Language") OR ("Domain Specific Modeling Language") OR ("Domain-Specific Modeling Language") OR ("Domain-Specific-Modeling-Language") OR ("Domain-Specific Modeling-Language"OR "Domain Specific Modeling-Language") WN KY) AND (tool) OR ("Language Workbench") WN KY) AND ((English) WN LA)
IEEE Xplore - DSL	(((((DSL OR "Domain Specific Language"OR "Domain-Specific Language"OR "Domain-Specific-Language"OR "Domain Specific-Language"OR "Small Language"OR "Small-Language"OR "Little-Language"OR "Little Language") AND (Tool OR "Language Workbench")))))
IEEE Xplore - DSML	(((DSML OR "Domain-Specific Modeling Language"OR "Domain Specific Modeling Language"OR "Domain-Specific Modeling-Language"OR "Domain Specific Modeling-Language") AND (Tool OR "Language Workbench")))
Science Direct	TITLE-ABSTR-KEY(DSL or DSML or "Domain Specific Language"or "Domain-Specific Language"or "Domain-Specific-Language"or "Domain Specific-Language"or "Little-Language"or "Little Language"or "Small-Language"or "Small Language"or "Domain Specific Modeling Language"or "Domain-Specific Modeling Language"or "Domain-Specific-Modeling-Language"or "Domain-Specific Modeling-Language") and TITLE-ABSTR-KEY(tool or "Language Workbench")
SCOPUS	(TITLE-ABS-KEY (dsl OR dsml OR "domain specific language"OR "domain-specific language"OR "domain-specific-language"OR "domain specific language"OR "domain specific modeling language"OR "domain-specific modeling language"OR "domain specific modeling-language"OR "domain-specific-modeling-language") AND TITLE-ABS-KEY (tool OR "language workbench")) AND (LIMIT-TO (LANGUAGE , "English"))
Springer Link	((DSL OR DSML "domain specific language"OR "domain-specific language"OR"domain-specific-language"OR ("Small Language") OR ("Small-Language") OR ("Little-Language") OR ("Little Language") OR ("Domain Specific Modeling Language") OR ("Domain-Specific Modeling Language") OR ("Domain-Specific-Modeling-Language") OR ("Domain-Specific Modeling-Language"OR "Domain Specific Modeling-Language") AND (tool or language workbench))

A faixa de anos pesquisado (2012 - 2017) não é exibida nas strings pelo fato de ter sido usado os filtros dos motores de busca.

Tabela 3 Pontuação da Qualidade dos Trabalhos

Trabalhos		CQ					Trabalhos		CQ				
ID	Referências	1	2	3	4	PF	ID	Referências	1	2	3	4	PF
S01	Demirkol et al. (2013)	1.0	1.0	0.0	0.0	2.0	S56	Jafer, Chhaya e Durak (2017)	0.5	1.0	1.0	0.5	3.0
S02	Mayr-Dorn e Laaber (2017)	1.0	1.0	1.0	0.0	3.0	S57	Zikra (2012)	1.0	1.0	0.0	0.5	2.5
S03	Visic et al. (2015)	0.5	1.0	0.0	0.0	1.5	S58	Ratiu e Ulrich (2017)	1.0	1.0	0.0	0.0	2.0
S04	Heitkötter (2012)	1.0	1.0	0.0	0.0	2.0	S59	Åkesson e Hedin (2017)	1.0	0.5	1.0	1.0	3.5
S05	Pomante, Candia e Incerto (2015)	1.0	1.0	1.0	0.0	3.0	S60	Wachsmuth, Konat e Visser (2014b)	1.0	1.0	0.5	0.0	2.5
S06	Barišić et al. (2017)	0.5	1.0	1.0	0.5	3.0	S61	Combemale, Barais e Wortmann (2017)	1.0	0.5	0.0	0.0	1.5
S07	Bermudez et al. (2017)	0.0	0.5	0.0	0.0	0.5	S62	Voelter et al. (2017)	0.5	1.0	1.0	1.0	3.5
S08	Sousa e Silva (2016)	1.0	1.0	0.0	0.0	2.0	S63	Alvarez e Casallas (2013)	1.0	1.0	0.5	0.0	2.0
S09	Hoyos, Garcia-Molina e Botia (2013)	1.0	1.0	0.0	0.0	2.0	S64	Vissers et al. (2017)	0.5	0.5	0.5	0.5	2.0
S10	Nazir et al. (2017)	0.5	0.0	0.5	0.0	1.0	S65	Jézéquel et al. (2015)	1.0	1.0	0.5	0.0	2.5
S11	Antonelli, Silva e Fortes (2015)	1.0	1.0	0.0	0.0	2.0	S66	Voelter et al. (2012)	1.0	1.0	0.0	0.0	2.0
S12	HoseinDoost et al. (2017)	1.0	0.5	1.0	0.5	3.0	S67	Molina et al. (2013)	1.0	1.0	0.0	0.0	2.0
S13	Hasan et al. (2017)	0.5	1.0	1.0	0.5	3.0	S68	Ma e Sallai (2017)	1.0	1.0	0.5	0.5	3.0
S14	Dejanović et al. (2017)	1.0	1.0	1.0	0.5	3.5	S69	Zhou et al. (2017)	0.5	0.5	0.5	0.0	1.5
S15	Gibbs, Dascalu e Harris Jr. (2015)	1.0	1.0	0.5	0.0	2.5	S70	Vieira e Carvalho (2017)	1.0	1.0	1.0	0.5	3.5
S16	Mavropoulos et al. (2017)	0.5	1.0	0.0	0.0	0.5	S71	Nordmann, Wrede e Steil (2015)	1.0	1.0	0.5	0.0	2.0
S17	Dwarakanath et al. (2017)	1.0	1.0	1.0	0.5	3.5	S72	Falkner et al. (2013)	1.0	1.0	0.5	0.0	2.0
S18	Szabo et al. (2016)	1.0	1.0	0.5	0.0	2.5	S73	Vinogradov, Ozhigin e Ratiu (2015)	1.0	1.0	0.5	0.0	2.0
S19	Sandobalin, Insfran e Abrahao (2017)	0.5	0.5	0.5	0.0	1.5	S74	Wigand et al. (2017)	0.5	0.5	0.5	0.5	2.0
S20	Häser, Felderer e Breu (2016)	1.0	1.0	0.0	0.0	2.0	S75	Maróti et al. (2014)	1.0	1.0	0.5	0.0	2.0
S21	Bergenti (2014)	1.0	1.0	0.0	0.0	2.0	S76	Bonnet et al. (2016)	1.0	1.0	0.5	0.0	2.0
S22	Lemazurier, Chapurlat e Grossetête (2017)	0.5	0.0	0.5	0.0	1.0	S77	Maro et al. (2015)	1.0	1.0	0.5	0.0	2.5
S23	Walter, Parreiras e Staab (2014)	1.0	1.0	0.5	0.0	2.5	S78	Oliveira e Belo (2017)	0.5	0.5	0.5	0.5	2.0
S24	Córdoba-Sánchez e Lara (2016)	1.0	1.0	1.0	0.5	3.5	S79	Challenger et al. (2014)	1.0	1.0	0.0	0.0	2.0
S25	Huang et al. (2015)	1.0	1.0	0.0	0.0	2.0	S80	Korenkov, Loginov e Lazdin (2015b)	1.0	1.0	0.0	0.0	2.0
S26	Neubauer et al. (2017)	0.5	1.0	0.0	0.0	1.5	S81	Pescador et al. (2015)	1.0	1.0	0.0	0.0	2.0
S27	Ratiu e Voelter (2016)	0.5	1.0	1.0	1.0	3.5	S82	Savić et al. (2014)	1.0	1.0	0.0	0.5	2.5
S28	Naujokat et al. (2017)	0.5	1.0	1.0	1.0	3.5	S83	Nakamura et al. (2012)	0.5	1.0	0.0	0.0	1.5
S29	Gargantini e Vavassori (2012)	1.0	1.0	0.0	0.5	2.5	S84	Arendt, Taentzer e Weber (2013)	1.0	1.0	0.0	0.5	2.5
S30	Bernardino, Zorzo e Rodrigues (2016)	1.0	1.0	0.5	0.5	3.0	S85	Ribeiro e Silva (2017)	1.0	1.0	0.0	0.0	2.0
S31	Hiya et al. (2013)	1.0	1.0	0.0	0.0	2.0	S86	Nagele e Hooman (2017)	0.5	1.0	0.0	0.0	1.5
S32	Stocker, Washizaki e Fukazawa (2017)	0.5	0.5	0.5	0.5	2.0	S87	Mohamad, Kolovos e Paige (2015)	0.5	1.0	0.0	0.0	1.5
S33	Zhu et al. (2017)	1.0	1.0	1.0	0.5	3.5	S88	Hoisl, Sobernig e Strembeck (2017)	0.5	0.5	0.0	0.0	1.0

Table 3 Continuação

Trabalhos		CQ					Trabalhos		CQ				
ID	Referências	1	2	3	4	PF	ID	Referências	1	2	3	4	PF
S34	Ribeiro, De Sousa e Da Silva (2016)	1.0	1.0	0.5	0.0	2.5	S89	Tikhonova (2017)	0.5	0.5	0.5	0.5	2.0
S35	Tairas e Cabot (2015)	1.0	0.0	0.0	0.0	1.0	S90	Monthe et al. (2016)	1.0	1.0	0.0	0.0	2.0
S36	Jager et al. (2016)	1.0	1.0	0.5	0.0	2.5	S91	Iliasov e Romanovsky (2013)	0.5	1.0	0.0	0.0	1.5
S37	Rocha et al. (2017)	1.0	1.0	0.0	0.0	2.0	S92	Haitzer e Zdun (2014)	1.0	1.0	0.5	0.0	2.5
S38	Pescador e Lara (2016)	1.0	1.0	0.5	0.5	3.0	S93	Viyović, Maksimović e Perišić (2014)	1.0	1.0	0.0	0.0	2.0
S39	Marand, Marand e Challenger (2015)	1.0	1.0	1.0	0.5	3.5	S94	Bartman et al. (2017)	0.0	0.5	0.0	0.0	0.5
S40	Tariq, Florence e Wolf (2014)	1.0	1.0	0.5	0.0	2.5	S95	Guelfi, Jahić e Ries (2017)	1.0	1.0	0.5	0.5	3.0
S41	Zarrin e Baumeister (2014)	0.5	0.5	0.0	0.0	1.0	S96	Uhnák e Pergl (2016)	1.0	1.0	1.0	0.0	3.5
S42	Bettini (2014)	1.0	1.0	0.5	0.0	2.5	S97	Erdweg et al. (2013a)	1.0	1.0	0.0	0.0	2.0
S43	Montrieux, Yu e Wermelinger (2013)	1.0	0.5	0.0	0.0	1.5	S98	Chlipala et al. (2017)	1.0	1.0	0.0	0.0	2.0
S44	Montenegro-Marin et al. (2012)	1.0	1.0	1.0	0.5	3.5	S99	Kowalski e Magott (2012)	0.0	0.5	0.0	0.5	1.0
S45	Arkin e Tekinerdogan (2014)	1.0	0.0	0.0	0.0	1.0	S100	Rozen et al. (2017)	1.0	1.0	0.0	0.0	2.0
S46	Krasts, Kleins e Teilans (2012)	1.0	0.0	0.0	0.0	1.0	S101	Granchelli et al. (2017)	1.0	1.0	0.5	0.5	3.0
S47	Jacob et al. (2014)	1.0	1.0	1.0	0.5	3.5	S102	De Faveri et al. (2017)	0.5	0.0	0.5	0.0	1.0
S48	Viana, Penteado e Prado (2013)	1.0	1.0	0.5	0.5	3.0	S103	Barišić, Amaral e Goulão (2017)	1.0	1.0	0.0	0.5	2.5
S49	Herrera (2014)	1.0	1.0	0.5	0.0	2.5	S104	Akhundov et al. (2016)	1.0	1.0	0.5	1.0	3.5
S50	Rose, Kolovos e Paige (2012)	1.0	1.0	0.5	1.0	3.5	S105	Hinkel et al. (2017)	1.0	1.0	0.0	0.0	2.0
S51	Erdweg et al. (2015b)	1.0	1.0	0.0	0.0	2.0	S106	Ribeiro e Da Silva (2014)	1.0	0.0	0.0	0.0	1.0
S52	Sutii, Brand e Verhoeff (2017)	1.0	1.0	0.0	0.0	2.0	S107	Efftinge et al. (2012)	1.0	1.0	0.5	0.0	2.5
S53	Viana et al. (2013)	0.5	1.0	0.5	0.0	2.0	S108	Santos e Gomes (2016)	1.0	1.0	0.0	0.0	2.0
S54	Smits e Visser (2017)	1.0	1.0	0.5	0.0	2.5	S109	Le Goaer e Waltham (2013)	0.5	1.0	0.0	0.0	1.5
S55	Rein, Hirschfeld e Taeumel (2016)	1.0	1.0	0.0	0.0	2.0							

4.2.1 Esquema de Classificação

O esquema de classificação foi gerado baseado na atividade "interpretação de dados". Assim, os trabalhos foram tabulados conforme as notações ao qual apoiam o desenvolvimento e as suas licenças de distribuição. é possível ver uma descrição de cada tipo de informação das tabulações na Tabela 4

Tabela 4 – Esquema de classificação

Característica	**Tipo**	**Descrição**
Notação	Gráfica	O suporte à notação gráfica fornece apoio a definição de elementos gráficos e visuais que auxiliam a modelagem do domínio e expressividade de elementos através de representações gráficas.
	Textual	A notação textual fornece suporte a definição de gramáticas e estruturas textuais, fornecendo elementos para expressividade textual dos elementos do domínio.
Licença	Comercial	Ferramentas lançadas sob esta licença estão sujeitas a cobranças financeiras pelo seu uso e posterior distribuição.
	Open source	Ferramentas que podem ser ou não distribuídas de forma livre, sem custos, ou que seu código possa ser acessado sem restrições.

4.3 Resultados

Nesta seção serão discutidas as respostas das QPs do SMS.

QP1. *Quais são as ferramentas/Frameworks/Language Workbench usadas para o desenvolvimento de DSL?*

Após a análise dos cento e nove (109) trabalhos selecionados, foi identificado cinquenta e três (53) ferramentas de apoio ao desenvolvimento de *DSL*, como apresentado na Tabela 5.

É importante mencionar, que durante o estudo foi observado um grande número de citações para ferramentas como o Xtext, *EMF* e GMF, assim como a Papyrus e Sirius.

QP2. *Quais são as* notações (gráfica e/ou textual) que estas ferramentas dão suporte?*

Vinte e quatro (24) ferramentas foram mapeadas como sendo de suporte a notação gráfica (ver Tabela 5), representando 45,28% do total de ferramentas, sendo a notação mais utilizada. Vinte e uma (21) ferramentas foram mapeadas como de apoio a notação textual, representando 39,62% do total de ferramentas.

Algumas ferramentas como, MPS e Whole, fornecem suporte a todas as notações. No que diz respeito ao suporte as notações gráficas, tabulares e simbólicas, apenas a MetaEdit+ apresentou suporte a estas três notações juntas. As ferramentas Enso, GEMOC,

Rascal e Spray fornecem suporte a ambas notações, gráfica e textual, representando 7.55% do total de ferramentas.

É importante mencionar que nenhuma das ferramentas mapeadas fornece suporte a todas as notações. Algumas das ferramentas fornecem suporte a apenas uma atividade de desenvolvimento de *DSL*. Outras como, JET, Acceleo e xpand, são geradores de código ou dialetos de outras linguagens como JAVA.

QP3. *Sob quais licenças estas ferramentas foram lançadas?*

Neste SMSs os estudos foram classificados em duas categorias conforme suas licenças: comercial ou *open source* (ver Tabela 5). As informações foram retiradas dos estudos, mas em alguns casos, por omissão ou falta da informação, foi necessário buscar estas informações nos sites oficiais, repositórios e desenvolvedores das ferramentas.

Trinta e nove ferramentas foram identificadas como *open source*, enquanto sete (7) foram identificadas como de licença comercial.

Entretanto, algumas ferramentas não puderam ter suas licenças mapeadas, no total de sete (7), pois estas ferramentas não foram oficialmente lançadas e não há informações sobre elas em buscas online[8].

[8] As ferramentas foram pesquisadas no Google (<www.google.com>)

Tabela 5 – Ferramentas, Notações e Licenças.

Ferramenta	**Notação**	**Licença**
ADOxx	Gráfica	Open Source
Acceleo	Textual	Open Source
Argyle	Textual	N/A
Clooca	Gráfica	Open Source
DEViL	Gráfica	Open Source
DSL-tao	Gráfica	Open Source
DrRacket	Textual	Open Source
EMF	Gráfica	Open Source
EMFText	Textual	Open Source
Eco	Textual	N/A
Edapt	Textual	Open Source
Enso	Gráfica e Textual	Open Source
Enterprise Architect	Gráfica	Comercial
Epsilon	Textual	Open Source
EuGENia Live	Gráfica	Open Source
GEF	Gráfica	Open Source
GEMOC	Gráfica e Textual	Open Source
GME	Gráfica	Open Source
GMF	Gráfica	Open Source
GMP	Gráfica	Open Source
Gramada	Textual	N/A
Graphiti	Gráfica	Open Source
JET	Textual	Open Source
Kermeta	Textual	Open Source
Kitalpha	Gráfica	Open Source
MPS	Gráfica e Textual	Comercial
Marama	Gráfica	Open Source
Melange	Gráfica e Textual	Open Source
MetaEdit+	Gráfica	Comercial
Microsoft DSL tools	Gráfica	Comercial
Microsoft Visual Studio	Gráfica	Comercial
Monticore	Textual	Open Source
Más	Gráfica e Textual	N/A
Nitra	Textual	Comercial
Obeo Designer	Gráfica	Comercial
Onion	Textual	N/A
Papyrus	Gráfica	Open Source
RSA	Gráfica	Open Source
Rascal	Gráfica e Textual	Open Source
Sirius	Gráfica	Open Source
Spoofax	Textual	Open Source
Spray	Gráfica e Textual	Open Source
SugarJ	Textual	Open Source
Tau G2	Gráfica	Comercial
textX	Textual	Open Source
Tool with no name	Textual	N/A
VL-Eli system	Gráfica	N/A
WebGME	Gráfica	Open Source
Whole	Gráfica e Textual	Open Source
XMF-Mosaic	Gráfica	Open Source
Xbase	Textual	Open Source
Xpand	Textual	Open Source
Xtext	Textual	Open Source

Fonte: Autor

5 LINGUAGEM ESPECÍFICA DE DOMÍNIO PARA TESTES DE DESEMPENHO EM SISTEMAS WEB

Neste capítulo será apresentado a implementação da DSL OpenMLPerf. Serão discutidos aspectos que vão desde a definição dos requisitos e decisões de design, até a implementação da linguagem com o LW Sirius. Na Seção 5.1 são discutidos os requisitos modelados conforme o domínio de teste de desempenho e as decisões de design. A Seção 5.2 apresenta o metamodelo e arquitetura base da DSL. A Seção 5.3 mostra a implementação da linguagem com o framework Sirius. Por fim, a Seção 7.1 discute as lições aprendidas com o desenvolvimento da DSL.

5.1 Requisitos e decisões de design

A análise de domínio e juntamente com a sua formalização, através de uma ontologia, foram definidas no trabalho de Bernardino (2016). A ontologia fornece uma base para a determinação de conceitos e relações que representam o domínio. Apesar da base de requisitos também ter sido criada neste trabalho, melhorias e mudanças foram realizadas. As decisões de design foram baseadas nas necessidades impostas por estes requisitos. Nesta seção serão apresentados os novos requisitos e decisões de design para a *DSL*.

5.1.1 Requisitos da Linguagem

Esta seção enumera os novos requisitos contemplados para a proposta da OMLPerf. A base dos requisitos foi previamente estabelecida no estudo de Análise de Requisitos e Decisões de Projeto para a OpenMLPerf (BERNARDINO, 2016).

Requisito (RQ)1) *A linguagem deve ser implementada e disponibilizada sob uma licença open source.*

Como principal mudança em relação a primeira versão da OpenMLPerf, qualquer ferramenta utilizada na implementação da linguagem deve ser *open source*, permitindo que ela seja disponibilizada em repositórios abertos a comunidade, academia e indústria.

RQ2) *A DSL deve prover uma representação gráfica das características do teste de desempenho.*

Esse requisito não diz respeito a linguagem, mas sim a ferramenta utilizada para sua criação. Dessa forma, a ferramenta deve ser capaz de fornecer mecanismos de representar graficamente todas as características do domínio.

RQ3) *A DSL deve prover suporte a uma representação textual.*

Dessa forma, engenheiros de testes, que estão acostumados com representação textual, irão ter mais facilidade em adotar a DSL, bem como documentar suas aplicações. A linguagem deve ter características e palavras-chaves que lembram o domínio de teste de desempenho. Este requisito se aplica para a integração da *DSL* com a linguagem textual proposta em um outro trabalho de conclusão de curso proposto por um aluno da

Universidade Federal do Pampa (UNIPAMPA).

RQ4) *A DSL deve permitir a rastreabilidade entre os elementos das notações gráficos e textuais.*

5.1.2 Decisões de Projeto

Esta seção descreve as decisões de projeto para a criação da nova versão da linguagem, relacionadas aos requisitos mencionados na Seção 5.1.1. Assim, para cada decisão será apresentado seu(s) respectivo(s) requisito(s).

Decisão de Design (DD)1 *O uso de soluções open source no auxílio da implementação de DSL gráficas* (RQ1, RQ2).

Estes requisitos foram atendidos por meio de um mapeamento da literatura realizado, apresentado no Capítulo 4, onde foram encontradas algumas ferramentas que atendiam a este requisito. Sendo assim, foi decidido pelo uso dos *frameworks* EMF e Eclipse Sirius, de código aberto sob as licenças Eclipse Public License 2.0 e Eclipse Public License 1.0, respectivamente e que fornecem suporte a criação de *DSL* com notação gráfica.

DD2) *Permitir suporte e rastreabilidade e entre as notações gráficas e textuais* (RQ2, RQ3, RQ4).

Este requisito foi satisfeito com a escolha das ferramentas de desenvolvimento do Eclispe para *DSLs*, EMF e Sirius, para futuramente permitir a integração da notação gráfica com a implementação da notação textual para a *DSL*. Como não existe uma única ferramenta *open source* que atenda a isso, será realizada uma integração entre os *frameworks* Sirius (Eclipse Foundation, 2018b) e Xtext (EFFTINGE et al., 2012). A LW deve permitir a conversão/tradução entre as regras dos modelos. Ou seja, a modelagem bidirecional entre os elementos gráficos e seus respectivos ativos na notação textual devem ser mapeadas. Não sendo este mapeamento apenas de um para um, mas que permita que uma notação gráfica seja mapeada para várias instâncias textuais.

5.2 Arquitetura e modelagem do domínio com Eclipse Ecore

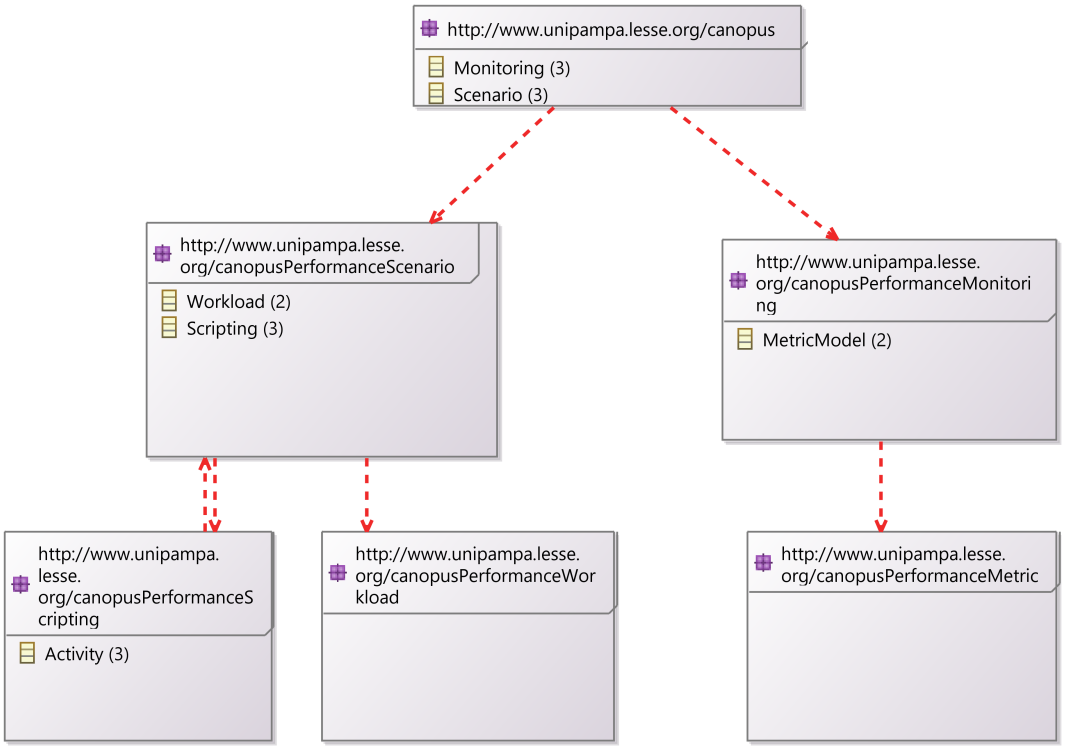
Nesta seção será apresentada a modelagem conceitual ou metamodelo do domínio de tese de desempenho. A modelagem foi realizada utilizando o framework EMF e seu metamodelo Ecore, através da ferramenta Ecore Tools, um *plugin* instalado diretamente no *marketplace* do Eclispe.

O metamodelo da linguagem foi concebido baseado nos requisitos e decisões de design apresentados na Seção 5.1. A linguagem é composta em duas partes: *Scenario*, *Monitoring* e *Scripting*. Estes conceitos foram abstraídos do domino de teste de desempenho.

Na Figura 10 é possível observar através de um diagrama de dependências como foi

definida a composição da linguagem, com dois metamodelos principais: O *OpenMLPerf Performance Monitoring*, *OpenMLPerf Performance Scenario*. Juntos, compõem o modelo principal da linguagem, chamado de Canopus. Somam-se a estes dois metamodelos, mais três metamodelos secundários, que fornecem suporte aos metamodelos principais, são eles: *OpenMLPerf Performance Metric*, *OpenMLPerf Performance Workload* e *OpenMLPerf Performance Scripting*.

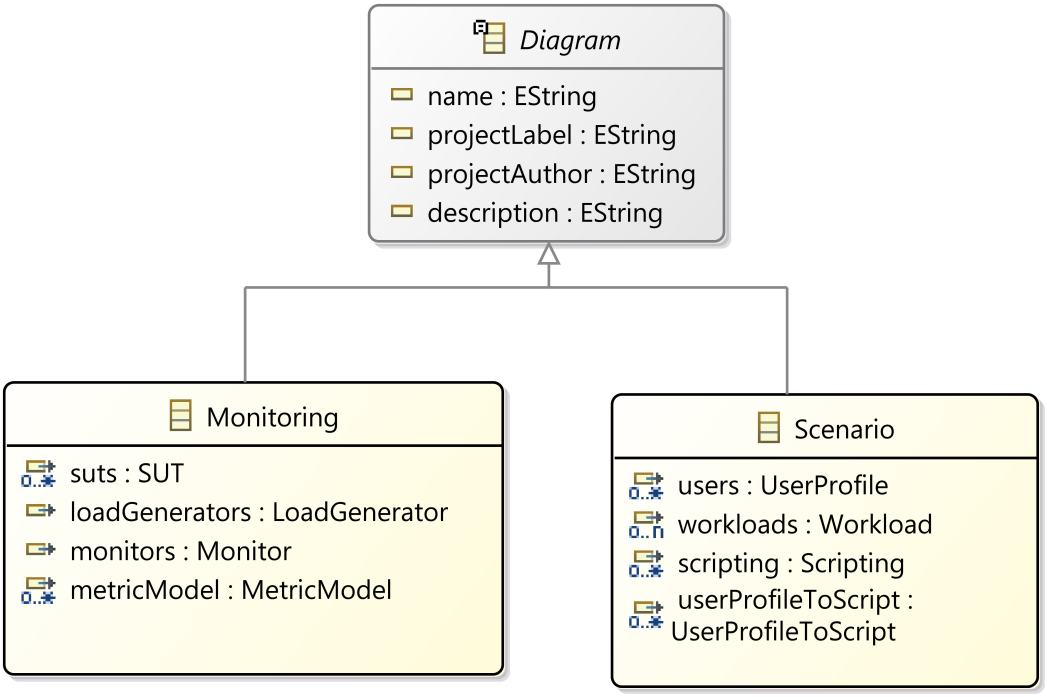
Figura 10 – Diagrama de dependência entre pacotes exibindo o modelo principal e os seis metamodelo da linguagem



Fonte: Autor

Os objetos do cenário e do monitoramento são instanciados dentro de um objeto do tipo canopus, que representa o projeto em si, ou seja, um projeto pode ter tanto um ou vários cenários, como pode ter um ou vários monitoramentos. Assim como é possível criar todos os objetos derivados, como *scriptns, workload e metrics*. Na Figura 11 é possível visualizar como são instanciados os modelos de *workload* e *script* no cenário, assim como o modelo de métrica dentro do modelo de monitoramento. Ambos os modelos, o de monitoramento e o de cenário, representam diagramas, onde podem ser definidos os elementos que compõem um teste de desempenho. Sendo assim, em cada modelo, são instanciados *SUTs*, monitores, geradores de carga e as métricas para o teste. Assim como, carga de trabalho, perfis de usuários e *scripts* de teste para um cenário de teste.

Figura 11 – Diagrama de classes do pacote OpenMLPerf



Fonte: Autor

OpenMLPerf Perfomance Monitoring Metamodel

O metamodelo *OpenMLPerf Perfomance Monitoring* (ver Figura 12) é responsável por definir os elementos responsáveis pelo ambiente de teste de desempenho. Nele estão representados os *SUT*, *i.e.*, aplicações *desktop* ou aplicados web, geradores de carga e as máquinas de monitoramento, ambas podem ser máquinas físicas, virtuais ou hospedadas em nuvem. É possível observar que existem dois tipos de enumeradores para definir o tipo do SUT e do *hardware*, essa abordagem se revelou bastante eficiente para definir elementos para escolha em lista, tanto para uma, como para múltiplas alternativas. Na Figura 12 é possível observar a associação possível entre vários SUTs, também é permitido que um *monitor* pode ser associado a vários SUTs para realizar o monitoramento. Um gerador de carga também pode ser associado tanto a vários SUTs, como tambem poder ser associado a um *monitor*.

Este metamodelo estabelece a necessidade de ao menos um SUT, um gerador de carga e um monitor para que seja completa a modelagem. O gerador de carga também pode ser responsável pelo monitoramento do cenário. O SUT deve ter um conjunto de métricas associadas que podem ser decompostas em um outro modelo de métricas. O modelo de métrica é definido pelo metamodelo *OpenMLPerf Performance Metric*.

O *OpenMLPerf Performance Metric* (ver Apêndice A) estabelece todos as métricas a serem monitoradas, suas relações e contadores. Nesse modelo são definidos os valores a

serem monitorados em cada uma das métricas associadas ao SUT.

OpenMLPerf Perfomance Scenario Metamodel

O metamodelo *OpenMLPerf Perfomance Scenario* (ver Figura 13) é responsável por definir os perfis de usuário e sua carga de trabalho. Nele ficam definidos ainda o fluxo dos *scripts* que representam uma atividade do usuário, como por exemplo, determinar a porcentagem de tempo em que um usuário irá pesquisar ou comprar em um site da internet. É necessário definir um dos perfis de carga de trabalho para decidir qual carga de trabalho será carregada no cenário. Este metamodelo possui uma relação com o *OpenMLPerf Perfomance Scripting Metamodel*, uma vez que os *scripts* que representam as atividades do usuário são modelados neste metamodelo.

O metamodelo *OpenMLPerf Performance Workload* (ver Apêndice A) define a estrutura das cargas de trabalho. Nele serão modeladas as quantidades de usuário no sistema, o aumento *(Ramp Up)* ou redução *(Ramp Down)* dessa população e o intervalo de tempo.

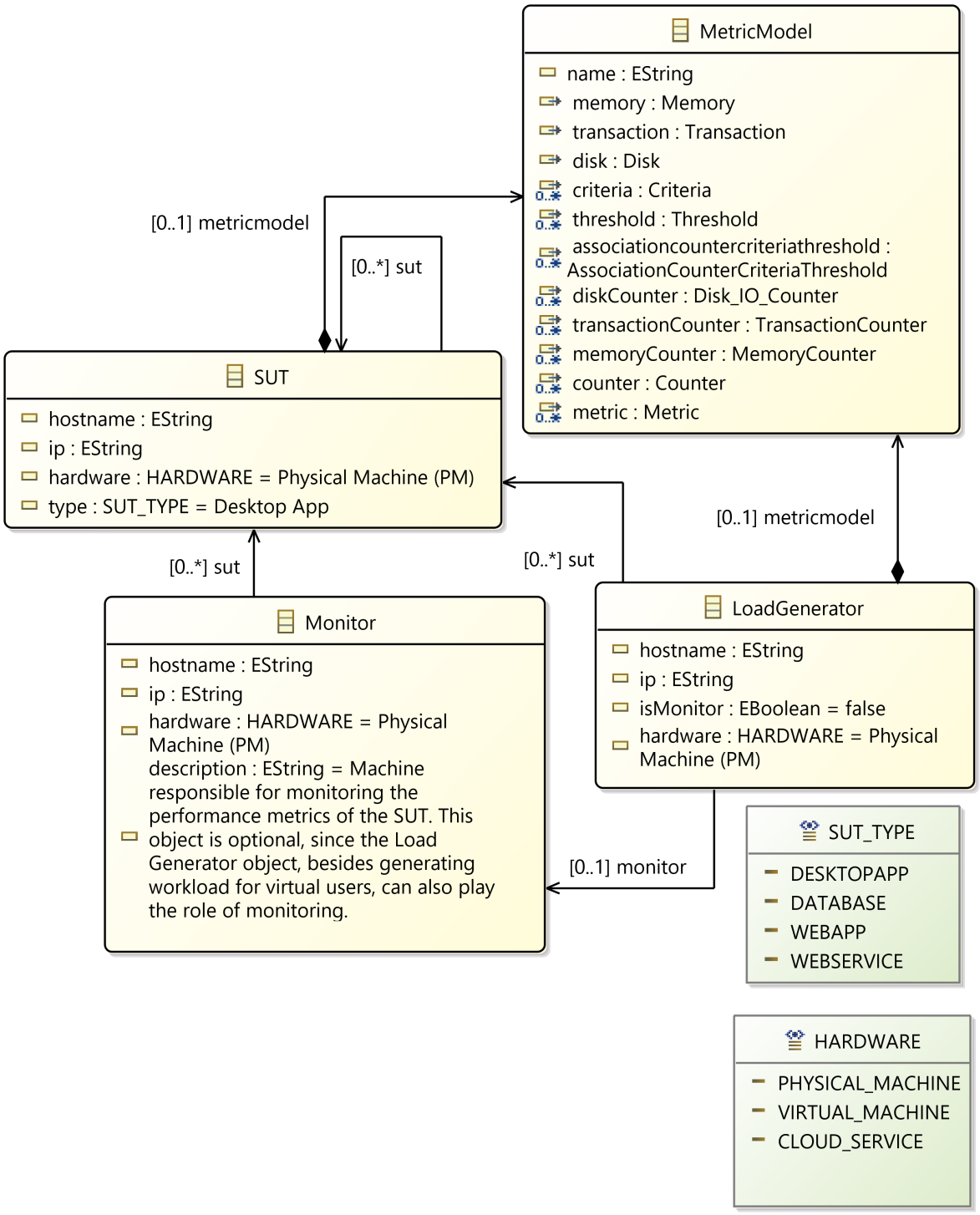
OpenMLPerf Perfomance Scripting Metamodel

Toda a modelagem dos scripts das tarefas e atividades realizadas pelo perfil do usuário são modeladas no metamodelo OpenMLPerf Perfomance Scripting (ver Figura 14). Neste metamodelo foi definido como devem ser as interações entre o perfil do usuário e os SUTs. Cada *script* é invariavelmente composto por um marco inicial e final. Além de atividades, também há elementos como, *Think Time*, que é o tempo entre a tarefa estar disponível para o usuário e o tempo da execução desta tarefa, *Data Table*, para recuperação e armazenamentos de informações para parametrização. É possível salvar os parâmetros para serem reaproveitados em outros modelos de *script*. Neste modelo também foi utilizado a estratégia de definir enumeradores para listas de características para várias propriedades do modelo. O metamodelo de apoio do *OpenMLPerf Perfomance Scripting*, o *OpenMLPerf Performance External File*, nada mais é, que um modelo para definir a persistência e recuperação de informações relevantes para os *scripts*.

5.3 Implementação da linguagem com o framework Eclipse Sirius

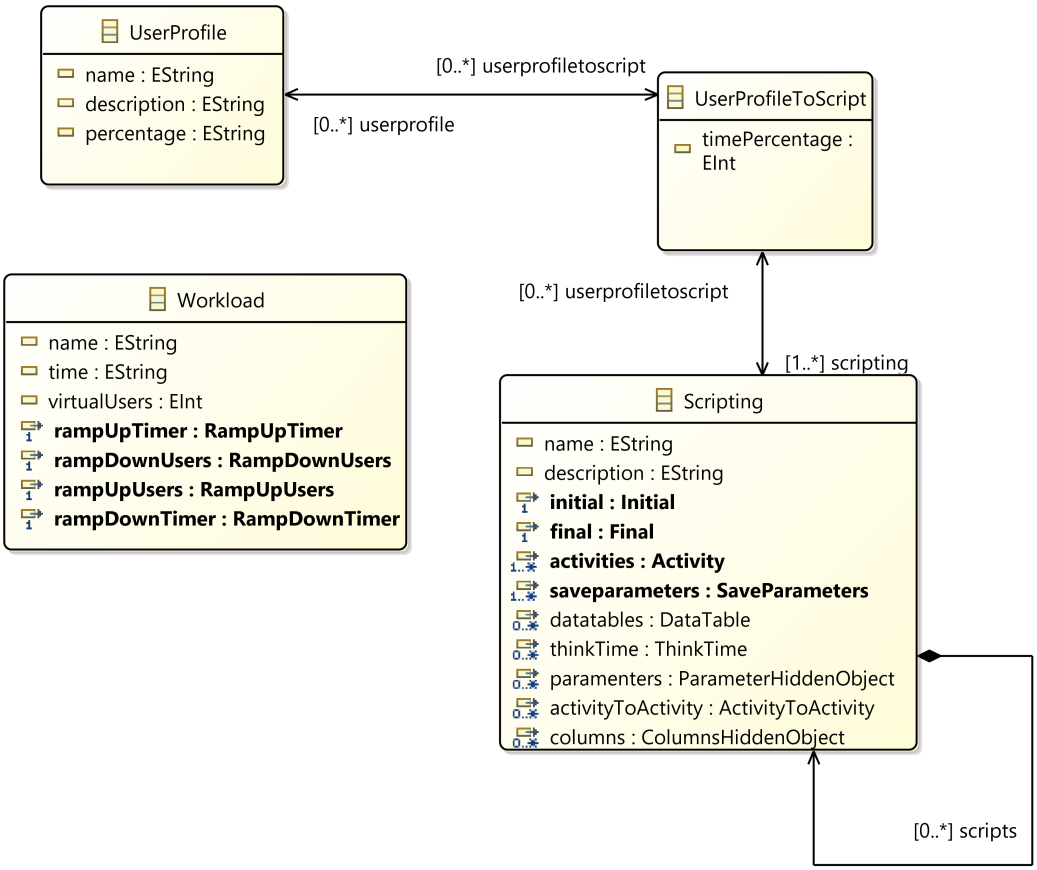
Nesta seção será apresentada a implementação da linguagem com o framework Sirius. Após a criação de um metamodelo com o framework EMF e o seu metamodelo Ecore, deve-se criar especificações para os elementos e propriedades do metamodelo, isto nada mais é do que criar representações gráficas da linguagem. São criadas especificações para tudo o que precisar ser representados, desde os elementos, no caso, diagramas, objetos, *i.e.*, monitores, SUTs, *scripts*, na forma de nós, até relações, *i.e.*, ligações entre SUTs e monitores e relações entre atividades em um *script*.

Figura 12 – Diagrama de classes exibindo o metamodelo de monitoramento da linguagem



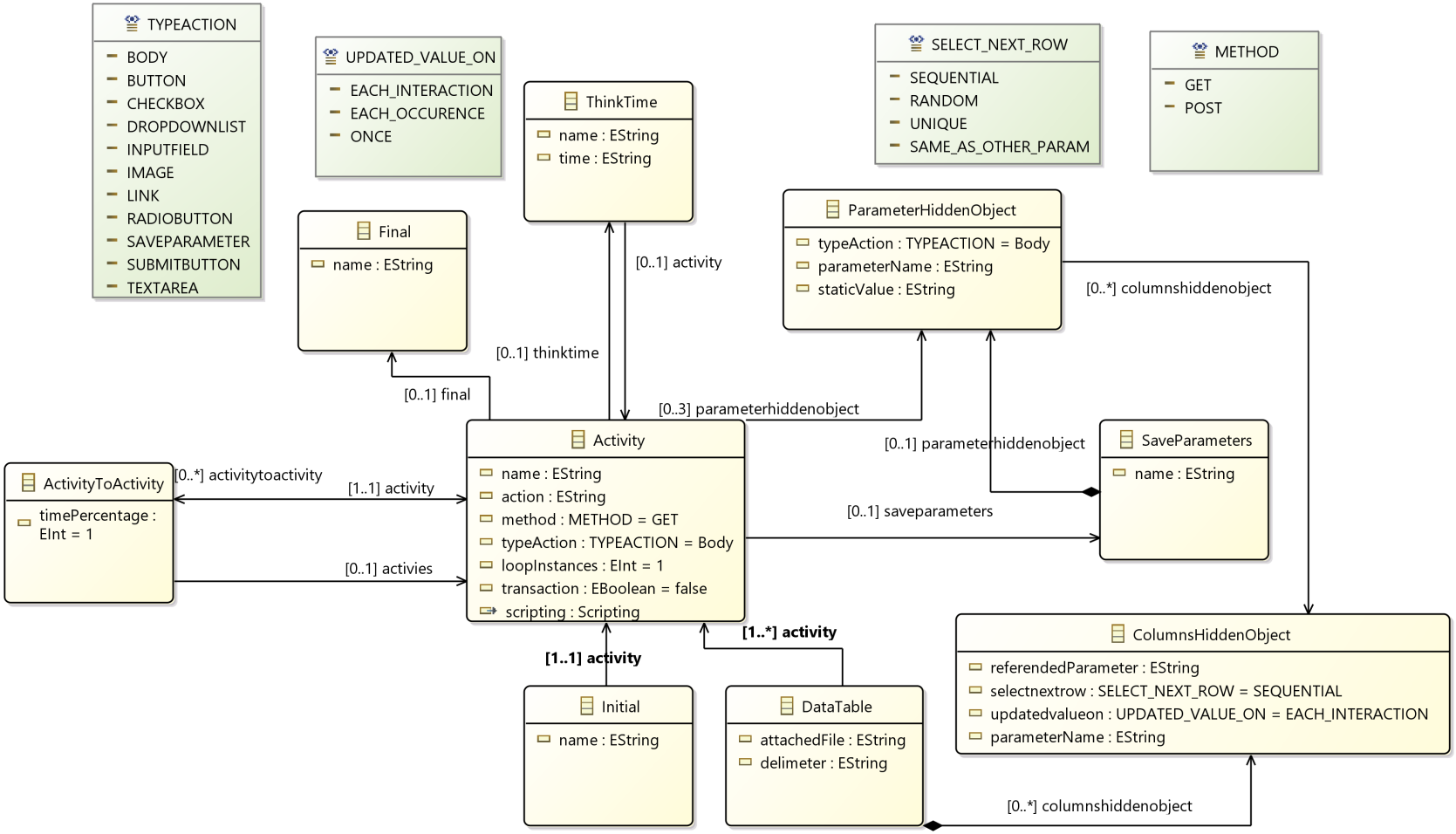
Fonte: Autor

Figura 13 – Diagrama de classes exibindo o metamodelo do cenário da linguagem Autor



Fonte: Autor

Figura 14 – Diagrama de classes exibindo o metamodelo de scripting da linguagem

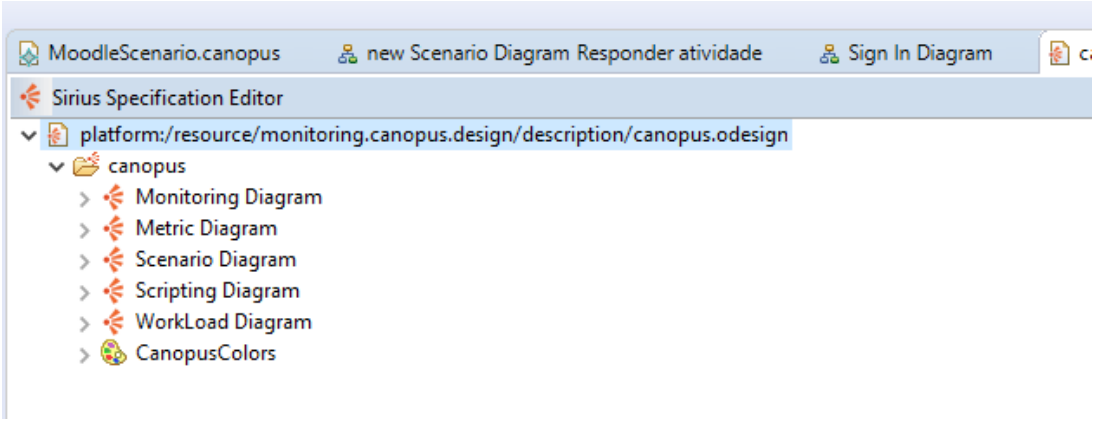


Fonte: Autor

Definição de Nós (*Nodes*)

Para cada modelo, baseado no metamodelo, que se deseja uma representação gráfica, existe uma representação em um *viewpoint specification* ou modelo de especificação no editor do Sirius. Como pode ser visto na Figura 15, foram criadas três *viewpoint specifications*, um para cada um dos modelos principais definidos no metamodelo da linguagem. Assim, cada um deles pode ser representado, editado e modelado dentro do framework de modelagem e posteriormente pelo usuário final.

Figura 15 – Editor de especificação do Sirius e os *viewpoint specification* relacionados aos modelos



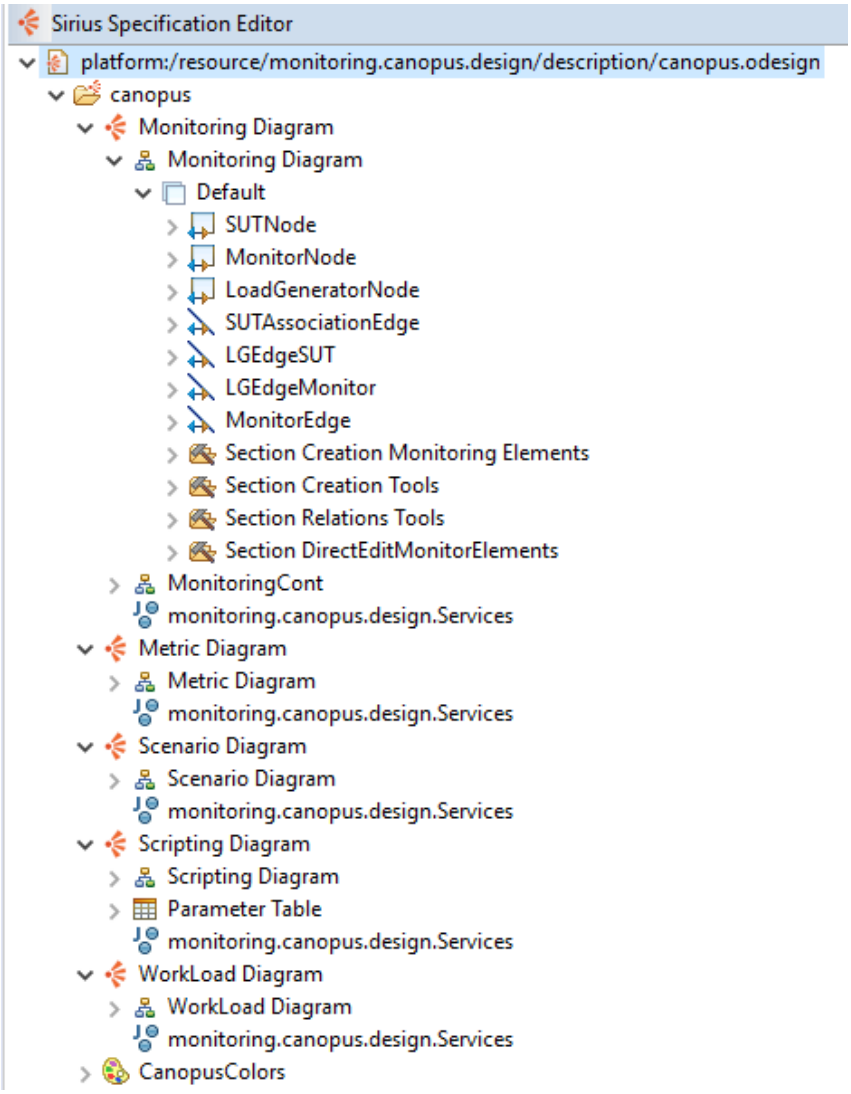
Fonte: Autor

Em cada *viewpoint specification* é possível criar inúmeras representações do mesmo modelo, chamadas de descrições, que podem ser diagramas, tabelas de edição, tabelas cruzadas, árvores e diagramas de sequência. Observa-se que na Figura 16 foi criado a descrição de um diagrama para representar o modelo de monitoramento (*Monitoring Diagram*).

O mesmo princípio é utilizado para criar uma representação para cada elemento do modelo que se deseja uma representação gráfica, para isso é criado um nó e atribuído a um elemento semântico do modelo. Na Figura 16 visualizam-se os três elementos principais do modelo de monitoramento, o SUTNode, representando o SUT nos diagramas de monitoramento, o MonitoringNode, que representa os servidores de monitoramento e o LoadGenerator, responsável por representar os servidores geradores de carga. Na definição dos nós é possível atribuir imagens e ícones ao invés dos ícones padrão do EMF. Alguns dos elementos do trabalho possuem ícones na representação, mas apenas como forma de testar esta funcionalidade, portanto, não sendo definitivos.

Na Figura 17 é apresentado o *viewpoint specification* para o modelo *OpenMLPerf Specification Metric*. Este modelo de métricas está associado a um SUT. Ao ser associado, expande-se o modelo em um diagrama para a modelagem dessas métricas, que também podem ser associadas a mais de um SUT ou modelo de monitoramento.

Figura 16 – Nós modelados para representar os elementos do metamodelo *OpenMLPerf Specification Monitoring*



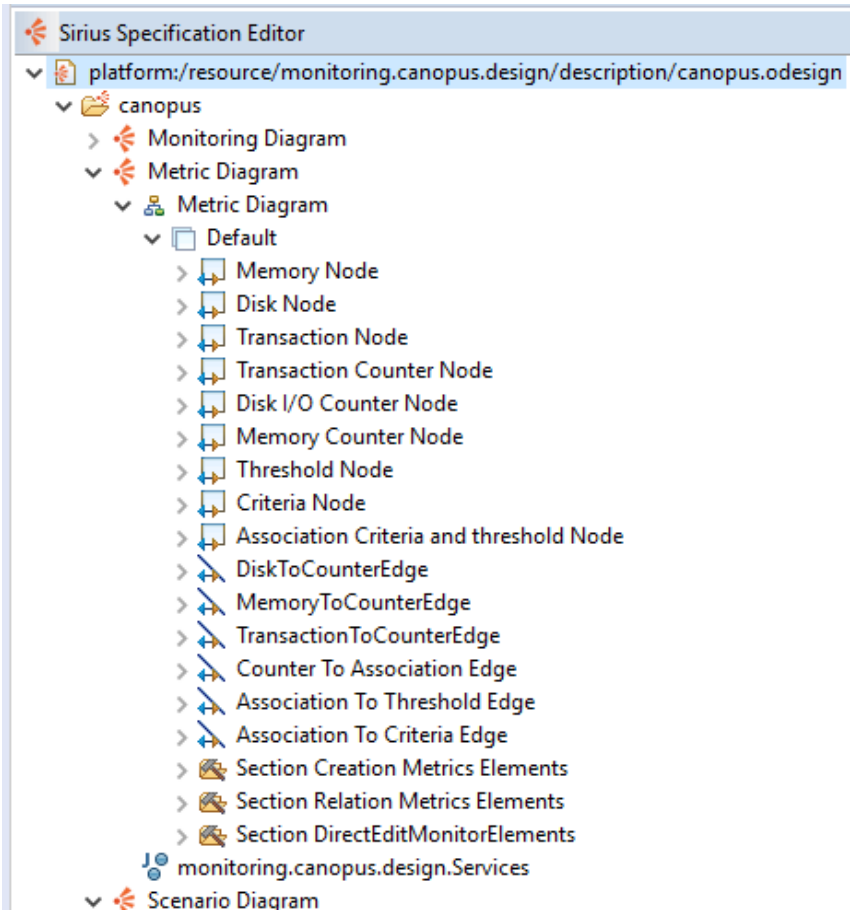
Fonte: Autor

5.3.1 Definição de relações

Para definir as relações, é criado um elemento do tipo *Edge* e atribuído a um elemento semântico de relacionamento entre classes do metamodelo. Então, se define a fonte do relacionamento *(source)* e o alvo *(target)*. Também se pode definir os tipos de seta, elemento da extremidade fonte e elemento da extremidade alvo. Neste trabalho foram definidos elementos de relação que expressassem o sentido de cada relação e diferencia-se uma relação da outra.

Na Figura 18 é possível ver a diferença entre os elementos das relações entre os SUTs e também entre o gerador de carga e os SUTs

Figura 17 – Nós modelados para representar os elementos do metamodelo *OpenMLPerf Specification Metric*



Fonte: Autor

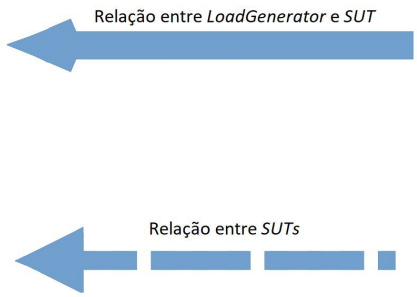
5.3.2 Diagramas para modelagem

A Figura 19 apresenta o diagrama de monitoramento, onde é possível especificar todas as informações relacionadas ao monitoramento do teste de desempenho, assim como a métrica associada ao elemento (caixa verde quadrada ao lado do elemento), tanto do *SUT*, quanto do gerador de carga. Também é possível definir identificadores, como, IP's, nomes e informações a respeito do tipo da máquina e do sistema, se físico, nuvem ou virtual.

Na Figura 20 podemos ver o diagrama de métricas do monitoramento, onde são definidas quais as métricas a serem monitoradas durante o teste. Neste diagrama é possível especificar a métrica, definir um ou mais contadores a serem monitorados, determinar os valores e intervalos a serem monitorados para cada métrica e criar critérios para a os valores da métrica.

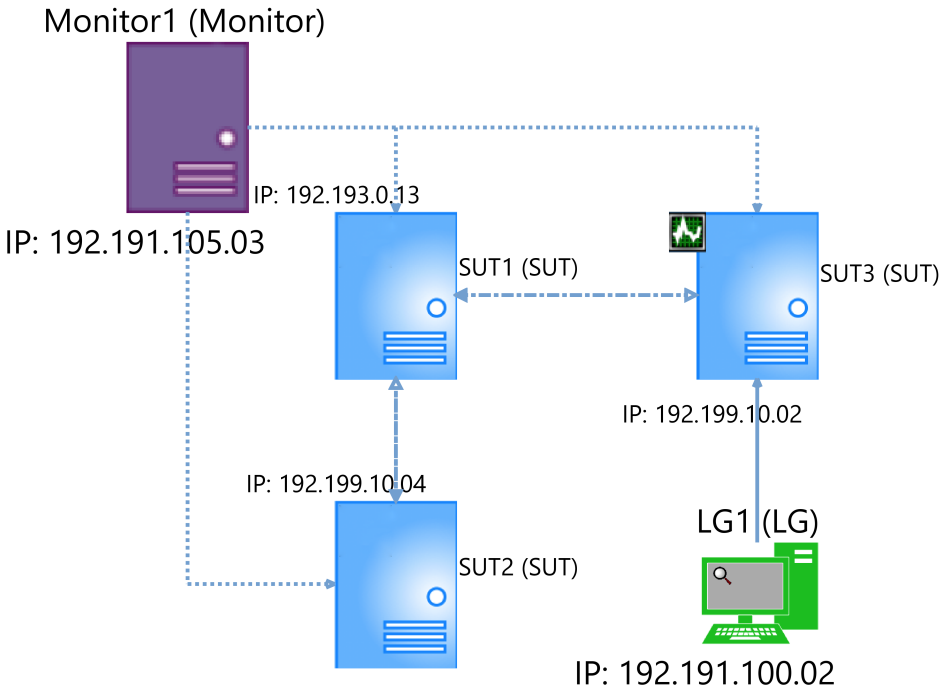
Como pode ser visto na Figura 21, o diagrama de modelagem de cenários permite, de forma visual, especificar um cenário de teste, com perfis de usuários, *scripts* e cargas de trabalho. Permite a definição de porcentagem de tempo a ser empregado por cada

Figura 18 – Exemplo de elementos gráficos de relação (*edges* entre objetos do modelo.)



Fonte: Autor

Figura 19 – Diagrama de modelagem do monitoramento do teste

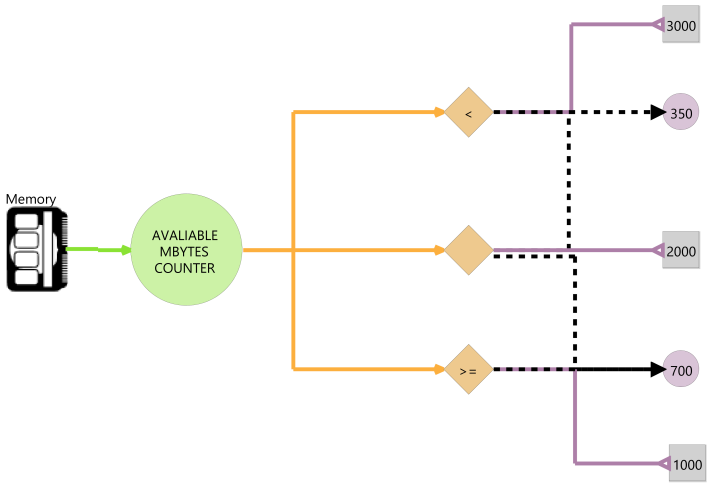


Fonte: Autor

perfil em uma determinada atividade e a reutilização dos perfis e *scripts* em mais de um modelo de cenário.

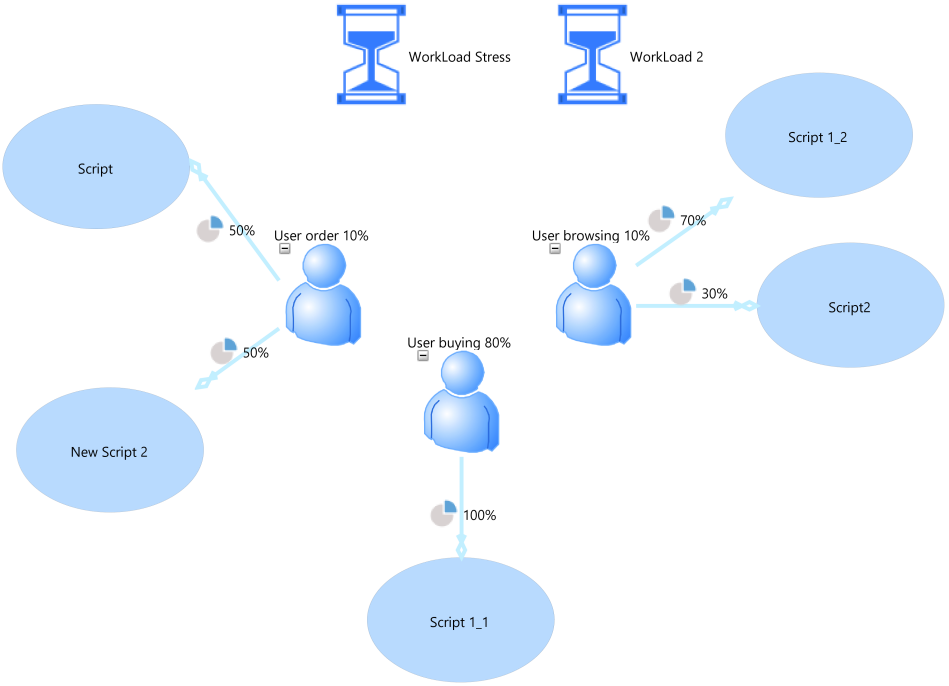
O diagrama para especificação de *scripts*, que pode ser visto na Figura 22, possibilita definir todas as atividades, que simularão a interação do usuário com o sistema, definir o tempo que irá levar para começar está atividade, a probabilidade de usuário realizar uma atividade ou outra, além de criar parâmetros para a atividade, *i.e.*, se a atividade será um clique em um botão, o preenchimento de um campo ou o carregamento

Figura 20 – Diagrama de modelagem da métrica do monitoramento



Fonte: Autor

Figura 21 – Diagrama de modelagem do cenário de teste

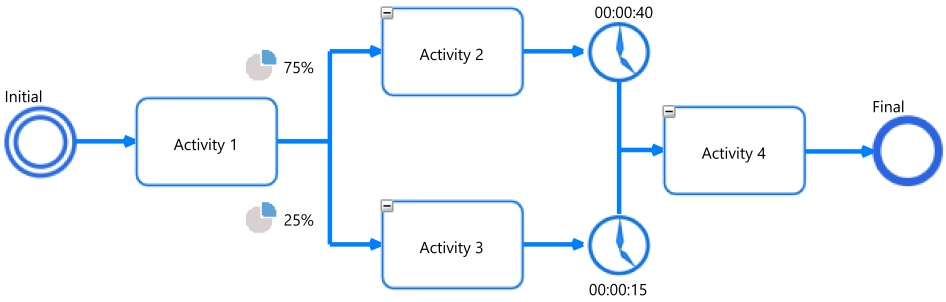


Fonte: Autor

de um formulário. Também é permitido armazenar um parâmetro, assim como um grupo de parâmetros, para que seja reutilizado em outras atividades e *scripts*.

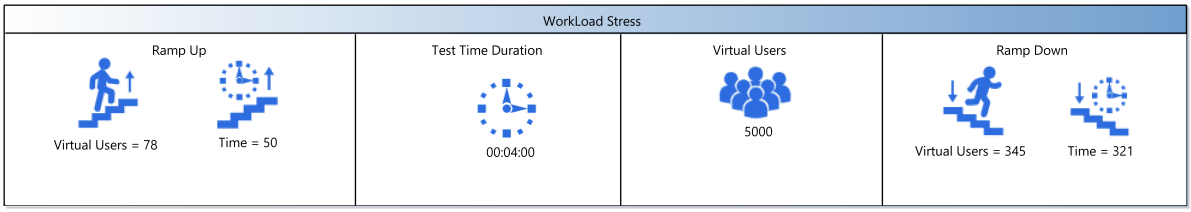
O diagrama de carga de trabalho permite definir a configuração das características de desempenho em um cenário de teste. Como pode ser visto na Figura 23, é possível definir a duração do teste, a quantidade de usuários virtuais no teste e a taxa de entrada e saída de usuários no sistema em um determinado intervalo de tempo.

Figura 22 – Diagrama de modelagem de um *script* do cenário



Fonte: Autor

Figura 23 – Diagrama de modelagem das cargas de trabalho do cenário



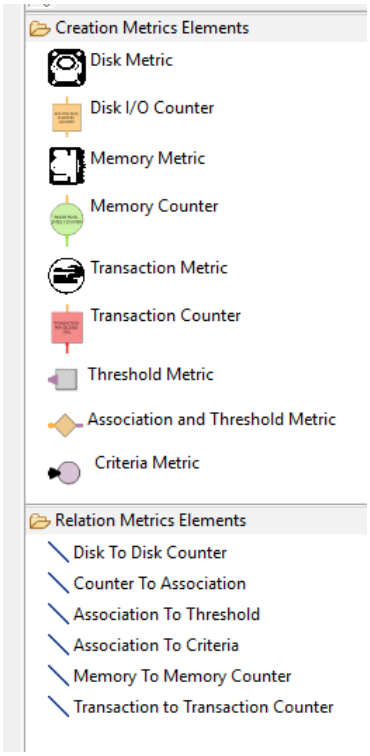
Fonte: Autor

5.3.3 Menus para modelagem

O framework Sirius também permite a especificação dos menus para serem utilizados pelo usuário na modelagem dos diagramas. O menu lateral funciona como uma paleta personalizada, onde é possível visualizar todos os elementos que podem ser criados no diagrama, assim como a todos as relações permitidas. Na Figura 24 e no Apêndice B é possível observar como estão definidos os menus da *DSL*. A especificação de menus inclui ainda a adição de menus contextuais, que aparecem junto ao ponteiro do mouse na do diagrama de modelagem. Nos menus contextuais é possível adicionar elementos a todos os diagramas do teste, assim como nos menus laterais. Apenas as representações de relações entre elementos não é possível realizar através dos menus contextuais, entretanto, além desta representação poder ser estabelecida pelos menus laterais, também pode ser estabelecida pelos menus de edição de propriedades.

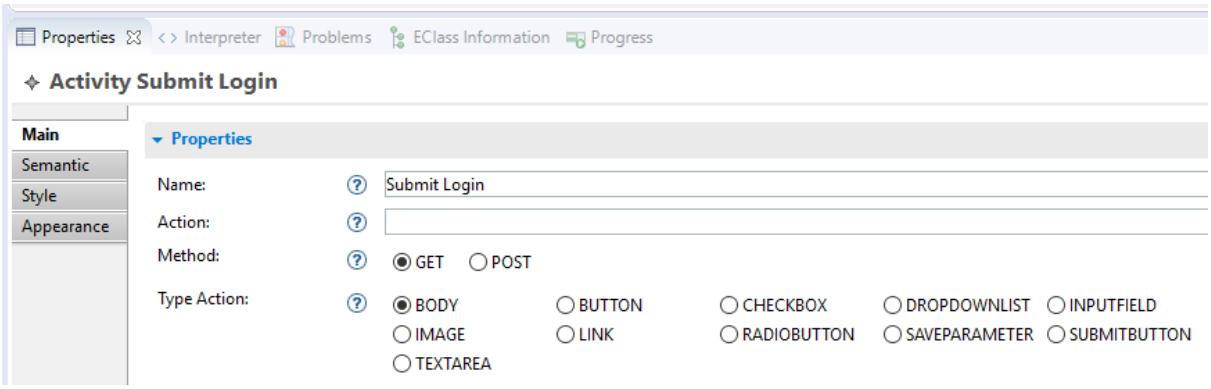
Os menus de edição de propriedades, como pode ser visto na Figura 25, permitem a edição de atributos relacionados aos elementos do diagrama e dos modelos associados, permitindo alterar componentes, editar aparência e estabelecer novas relações entre os diversos modelos de um mesmo metamodelo, desde de que estes já estejam previstos no metamodelo.

Figura 24 – Menu de criação do diagrama de *Scripting*



Fonte: Autor

Figura 25 – Menu de edição de propriedades



Fonte: Autor

5.3.4 Lições do capítulo

Neste capítulo apresentamos os requisitos e decisões de design para a OpenMLPerf. Discutimos cada requisito e apresentamos as decisões de design que o satisfizessem. Assim pudemos apresentar como foi realizada a implementação da linguagem. Foi mostrado como ficou estruturada a arquitetura da linguagem através dos diagramas de pacotes criados com o EMF. O EMF foi utilizado para criar o metamodelo da linguagem, no qual é definido cada aspecto estrutural, conceitual e de relacionamento que a linguagem

deve obedecer. No Capítulo seguinte será apresentada a avaliação executada para verificar o esforço para realizar a modelagem de teste de desempenho com a OpenMLPerf em comparação com um perfil UML para teste de desempenho. Serão apresentados e discutidos os resultados obtidos através da realização de uma modelagem e de um *survey* pós-experimento, onde os participantes responderam questões que envolviam a efetividade, facilidade e intuitividade da abordagem.

6 AVALIAÇÃO EMPÍRICA

Este Capítulo apresenta os resultados e discusões de uma avaliação empírica conduzida para avaliar o esforço em se utilizar a OpenMLPerf. Foram comparados os esforços de modelagem com a OpenMLPerf e o esforço de modelagem utilizando uma abordagem com modelos *UML* para teste de desempenho.

Esta avaliação empírica está organizada da seguinte maneira: A Seção 6.1 apresenta a definição do experimento, as questões de pesquisa e objetivos. Já a Seção 6.2 descreve o planejamento do experimento, suas hipóteses das questões de pesquisa e variáveis. Também é descrita a seleção dos participantes, o design da avaliação e as ameaças a validade da avaliação. Na Seção 6.3, os passos de Preparação e Execução realizados durante o experimento são discutidos. A Seção 6.4 apresenta os resultados encontrados na avaliação. Finalmente, a Seção 6.4.1 discute os resultados, a execução e melhorias da avaliação.

6.1 Definição da Avaliação

Nesta seção será apresentada as questões de pesquisa para a avaliação, a definição da avaliação e os objetivos de pesquisa da avaliação.Também serão apresentadas as hipóteses para as questões, bem como os instrumentos e o design da avaliação.

6.1.1 Questões de Pesquisa

O objetivo deste experimento é obter dados quantitativos ou qualitativos à respeito do esforço necessário para modelar cenários de teste de desempenho utilizando a OpenMLPerf e comparar com o esforço necessário ao realizar a mesma modelagem utilizando um perfil *UML* (RODRIGUES et al., 2015) para teste de desempenho. Criar cenários de testes e os testes em si, pode ser uma das tarefas mais demoradas durante o desenvolvimento de software. Utilizar atividades pertencentes ao *MDE*, como o *MBT* e *DSLs*, torna-se uma alternativa bastante promissora, tanto na economia de tempo, como custos. Uma vez que, este tipo de abordagem fornece a geração automática de *scripts* de teste após a modelagem dos cenários, referentes ao domínio a ser testado, sem a necessidade de entendimento das tecnologias responsáveis por executar os testes. Por outro lado, este tipo de abordagem exige um elevado nível de conhecimento acerca do domínio que será testado e do próprio domínio de teste, neste caso, o domínio de teste de desempenho.

Assim, foi decidido realizar este experimento para identificar informações sobre a efetividade, facilidade e esforço relacionados a modelagem de testes de desempenho, utilizando uma *DSL* e utilizando um perfil *UML*. Para isso, foi escolhido como estratégia de pesquisa, definir as questões baseadas nas tarefas que seriam designadas aos participantes da avaliação e que simulariam a modelagem de cenários em um ambiente normal da indústria, utilizando a OpenMLPerf e o perfil *UML* e assim, identificar o esforço necessário

para realizar as duas modelagens e comparar ambas as abordagens.

A avaliação contém as seguintes QPs a serem respondidas:

QP1. *Qual o esforço para modelar testes de desempenho utilizando um perfil UML e a OMLPerf?*

QP2. *Quão efetivo é modelar um teste de desempenho ao utilizar um perfil UML e a OpenMLPerf?*

QP3. *Quão intuitivo/fácil é modelar testes de desempenho utilizando um perfil UML e a OpenMLPerf?*

6.1.2 Definição do objetivo

Os objetos de pesquisa desta avaliação são medir o esforço e a intuitividade no uso da OpenMLPerf e do perfil *UML* para teste de desempenho. O esforço, medido em tempo, combinado com a intuitividade e facilidade, que é medida de acordo com as opiniões dos usuários das ferramentas, durante a modelagem de cenários de teste, ajudam a ter compreensão da praticidade nas construções de modelos e do tempo que é gasto para tal.

6.2 Planejamento

Nesta seção será apresentado o planejamento da avaliação, as questões de pesquisa e suas hipóteses. Também será apresentado como foram selecionados os participantes, a elaboração do experimento e as ameaças a validade.

6.2.1 Contexto

O contexto da avaliação pode ser separado em 4 conceitos:

- **Processo:** Foi utilizada uma abordagem in-vitro, em um ambiente semi-controlado, uma vez que as máquinas eram pessoais dos participantes, sem atividades online.
- **Participantes:** Os participantes são estudantes de graduação em cursos de computação.
- **Realidade:** A avaliação aborda problemas reais, como a modelagem de *scripts* de testes.
- **Generalidade:** Esta avaliação está inserida em um contexto especifico, mas os resultados podem ser utilizado para questões referentes a utilização de *DSLs* em ambientes diversos, *i.e.,* ambientes configurados para o uso das *DSLs*.

6.2.2 Formulação de hipóteses

Nesta seção serão apresentada as hipóteses e as medidas utilizadas para avaliar a resposta da QP1. Estas são as notações utilizadas para cada hipótese:

Φ_{dsl}: Representa a medida quando utilizada a OpenMLPerf para a modelagem de cenários de teste de desempenho.

Φ_{uml}: Representa a medida quando utilizada o perfil *UML* para a modelagem de cenários de teste de desempenho.

As questões de pesquisas, já apresentadas anteriormente, e suas hipóteses são:

QP1. *Qual o esforço para modelar testes de desempenho utilizando um perfil UML e a OpenMLPerf?*

Hipótese nula, H_0: $\Phi_{dsl} == \Phi_{uml}$: O esforço é o mesmo quando usando o o perfil *UML* e a OpenMLPerf para criar modelos de teste de desempenho.

Hipótese alternativa, H_1: $\Phi_{dsl} < \Phi_{uml}$: O esforço é menor quando usando o o perfil *UML* do quando usando a OpenMLPerf para criar modelos de teste de desempenho.

Hipótese alternativa, H_2: $\Phi_{dsl} > \Phi_{uml}$: O esforço maior quando usando o perfil *UML* do quando usando a OpenMLPerf para criar modelos de teste de desempenho.

6.2.3 Seleção dos participantes

Os participantes da avaliação são estudantes de graduação nos cursos de Engenharia de software e Ciências da Computação na *Universidade Federal do Pampa* (UNIPAMPA). Foram convidados os estudantes matriculados de forma regular sem litações quanto ao semestre atual no curso. Para fim de nivelamento de conhecimento, os estudantes tiveram que responder um *survey* (ver Apêndice C) com questões referentes ao nível de conhecimento do aluno em relação a técnicas de modelagem de software, tanto com *UML*, como que com *DSLs*. Também foram realizadas perguntas referentes ao nível de conhecimento técnico dos participantes em relação ao teste de desempenho e linguagens de modelagens. Como resultado deste *survey*, foi possível identificar um nível equilibrado de conhecimento dos participantes, não havendo nenhum participante fora da curva de conhecimento do grupo.

6.2.4 Design da avaliação

O design da avaliação segue as seguintes diretrizes:

Randomização: Os participantes foram distribuídos de forma aleatória para realizarem as tarefas com a OpenMLPerf e com o perfil *UML*.

Agrupamento: Como não houve diferença significativa entre os níveis de conhecimento dos participantes, os indivíduos foram separados em dois grupos homogêneos.

Nivelamento: Os participantes foram agrupados em dois grupos, em que cada grupo deveria realizar a tarefa com o uso de uma das abordagens.

6.2.5 Instrumentação

Para dar suporte aos participantes da avaliação, foram providenciados documentos com as especificações técnicas, casos de uso e requisitos de desempenho. Além disso, foram usadas duas ferramentas de apoio, uma para cada abordagem. Para a modelagem dos casos de uso e diagramas de atividades com a abordagem *UML* foi utilizada a ferramenta de modelagem Astah Professional. Para a modelagem dos diagramas de representação dos modelos da *DSL* foi utilizada uma versão adaptada da *IDE Eclipse* com suporte aos metamodelos da DSL.

Uma seção de treinamento foi executada, onde os conceitos relacionados a OpenML-Perf e ao perfil *UML* para teste de desempenho foram apresentados para os participantes. Um tutorial prático demonstrando como realizar a criação de modelos de desempenho com a ferramenta Astah e o perfil *UML* foi executado. Junto ao tutorial, foi entregue um manual com detalhes a respeito do perfil *UML* e instruções de como usar a ferramenta de modelagem Astah. Da mesma forma, um tutorial permitiu aos os participantes observarem a aplicação da OpenMLPerf para a criação de modelos de teste de desempenho. Também foi fornecido um manual com informações de como realizar a instalação dos modelos necessários para a execução da linguagem e de como realizar a modelagem de testes de desempenho.

Na sessão de execução do experimento, os participantes interagiram com uma aplicação web chamada Moodle. Para executar as tarefas de modelagem, foram utilizados os documentos, anteriormente mencionados, com as especificações técnicas dos testes descritas em detalhes. Assim, baseado nestes documentos, os participantes tiveram que criar os modelos de desempenhos.

O esforço de cada participante para realizar a tarefa de modelagem foi coletado, assim buscando responder a QP1. Para responder a QP1 e a QP2 os dados foram coletados em um *survey* realizado após o experimento.

6.2.6 Ameaças à Validade

Nesta seção serão descritas as ameaças a validade da avaliação e as ações de mitigação tomadas. O esquema de classificação apresentado por Hyman (1982) que classifica as ameaças em quatro tipos, foi adotado nesta avaliação. São essas as ameaças:

Validade da conclusão: O perfil dos participantes é uma ameaça significativa ao estudo, pois não houve representantes da indústria e o pequeno grupo de participantes da academia não possuía experiência considerável com linguagens modelagens, TD e perfis *UML*. O viés do pesquisador ao analisar os resultados também é uma ameaça a ser considerada, para isso, uma forma de mitigar a ameaça foi analisar de forma não humana o esforço, através de dados estatísticos de tempo. Outra ameaça foi a realização da avaliação em máquinas pessoais dos participantes e não em máquinas com a mesma configuração,

como em máquinas de um laboratório. Para mitigar, foi desenvolvido uma *build* única da *DSL* e disponibilizado uma mesma versão do Astah proffssional.

Validade interna: A seção do treinamento e a seção de avaliação foram realizadas em dias separados, o que é uma ameaça a validade interna do estudo. Uma vez que houve um intervalo de um dia entre as duas seções, o que pode acarretar esquecimento do que foi apresentado no treinamento. Para mitigar isso, foram distribuídos manuais e guias de cada abordagens. Para garantir que houvesse balanceamento entre os grupos de participantes, um *survey* foi aplicado, assim permitindo o agrupamento das amostras.

Validade externa: A seleção de participantes do experimento é uma das ameaças à avaliação, uma vez que não foi possível convidar participantes com experiencia considerável para a realização da avaliação.

Validade da construção: Uma ameaça a construção foi ter usado apenas uma aplicação e sistema para modelagem, limitando o número de variáveis.

6.3 Operação da Avaliação

Nesta seção serão discutidas as atividades de preparação e execução da operação da avaliação.

6.3.1 Preparação

A preparação do experimento começou com a escolha das aplicações e possíveis cenários para servirem como modelo para a criação dos cenários de testes de desempenho. Foram disponibilizados para todos os participantes os documentos de especificações, requisitos e casos de usos, os dos manuais das duas abordagens, além das versões executáveis das duas ferramentas utilizadas para modelagem, o *Astah* e o*Eclipse IDE*. Além disso, foi realizado um *survey* com o objetivo de obter informações e traçar um perfil dos participantes. Assim foi possível identificar que não havia discrepância entre os níveis de conhecimento dos participantes, se mostrando um grupo muito homogêneo.

6.3.2 Execução

A avaliação foi realizada no mês de março 2019. Uma seção de treinamento foi realizada nos dias anteriores a execução da avaliação, onde ambas as abordagens foram aplicadas, buscando assim passar uma visão geral de ambas as ferramentas e retirando dúvidas dos participantes. Para cada abordagem foi necessário realizar uma tarefa de modelagem, a de cenário de desempenho, que deveria ser expandida para a modelagem do *script* de desempenho.

Durante a seção de treinamento, os participantes precisaram modelar um cenário de desempenho que representa a interação entre dois perfis de usuários com o sistema de surveys online *LimeSurvey*: O perfil de estudante e o do profissional, que responderiam

Abordagem	Participantes
UML	5
OpenMLPerf	4

Tabela 6 – Número de participantes por abordagens.

um survey. O cenário continha uma carga de trabalho de mil usuários virtuais, uma duração de quatro horas de teste e executa um *script* denominado *"Responda o Survey"*, o qual é composto de 20 atividades representando a interação do usuário ao responder cada uma das questões do *survey*.

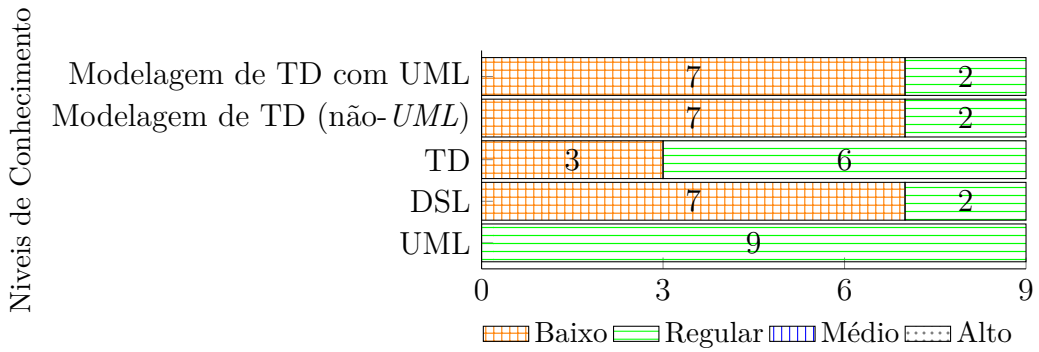
A tabela 6 apresenta a distribuição dos participantes na execução das abordagens. Nove participantes foram divididos nos dois grupos, onde cinco participantes executaram a abordagem com o perfil *UML* e 4 participantes realizaram as tarefas com a *DSL*

Na fase de execução os participantes realizaram uma tarefa utilizando a abordagem determinada para cada grupo. Cada tarefa pode ser descrita assim:

- UML: Os modelos de desempenho deveriam ser construídos de acordo com as especificações fornecidas, usando o perfil *UML* para teste de desempenho.
  - Modelagem do cenário: Nesta tarefa, cada participante construiu um modelo, baseado nos requisitos de desempenho, utilizando o Astah. Diagramas de caso de uso podiam representar o modelo de cenário de desempenho e os perfis de usuário, que representavam dois atores: estudantes e professores. Três casos de uso, que podiam ser representados com diagramas de atividades, poderiam ser associados aos usuários do cenário;
  - Modelagem do *script*: Para a realização da tarefa de *script*, os participantes poderiam se basear nas informações encontradas nas especificações do caso de uso fornecidas. Poderiam ser adicionadas informações relacionadas ao desempenho em cada atividade dentro do *script*.
- OpenMLPerf: Os modelos de desempenho deveriam ser construídos de acordo com as especificações fornecidas, usando a OpenMLPerf para teste de desempenho.
  - Modelagem do cenário: Para realizar esta tarefa, cada participante teve que usar a OpenMLPerf e construir **Modelos Cenário de Desempenho OpenMLPerf**. Para isso, foi utilizado como plataforma a IDE Eclipse, com adição dos *plugins* com os modelos e metamodelos necessários para a execução da DSL. Dentro de cada modelo de cenário era possível adicionar modelos de carga de trabalho, que continha informações de desempenho como, duração do teste e número de usuários virtuais.
  - Modelagem do *script*: Nesta tarefa, o **Modelo de Script de Desempenho OpenMLPerf** construído simularia a interação entre o usuário e o sistema.

Na Figura 26 é possível ver os resultados obtidos com a realização do *survey* para nivelamento de conhecimento, ao qual os participantes responderam previamente a execução da avaliação. É possível notar que todos os participantes mencionaram ter conhecimento regular de *UML* e que a maioria disse ter um baixo conhecimento sobre *DSL*, assim como em modelagem de TD com o uso de *UML* ou com outras linguagens de modelagem. A maioria dos participantes disse ter um conhecimento regular sobre TD. Importante mencionar que dos nove participantes que responderam ao *survey* de nivelamento de conhecimento, assinaram o termo de participação na avaliação e realizaram a seção treinamento, apenas 8 participantes participaram da execução da avaliação, já que um dos participantes se ausentou da execução. Também é necessário informar que um dos participantes solicitou para ser retirado do experimento, mesmo tendo realizado a tarefa de modelagem com o perfil *UML*, o participante não se sentiu a vontade com seus conhecimentos para ceder o seus modelos para a avaliação, assim sendo este participante foi descartado. Outro participante foi descartado devido a problemas técnicos com o equipamento do participante, o que acabou por corromper os modelos que o participante havia criado com o perfil *UML*. Estas dois descartados não tiveram seus valores de esforço inclusos nos resultados do experimento.

Figura 26 – Resultados das questões de nivelamento



6.4 Resultados

Nesta seção serão apresentados e discutidos os resultados encontrados durante a execução da avaliação empírica.

QP1. *Qual o esforço para modelar testes de desempenho utilizando um perfil UML e a OpenMLPerf?*

A tabela 7 apresenta um resumo dos dados relacionado ao esforço de cada participante para realizar a tarefa de modelagem, como descrito na Seção 6.3.2. Na coluna *"Tempo Médio"*, pode-se observar o esforço, medido como tempo médio, que cada participante levou para realiza as tarefas. Para realizar a tarefa com a abordagem *UML*, os participantes levaram em média 1h02min56s para modelar o cenário e o *script* pro-

Abordagem	Grupo	Tempo Médio (MM:SS)	Mediana
UML	1	63:15	63:15
OpenMLPerf	2	66:48	65:30

Tabela 7 – Esforço dos participantes (Média e Mediana)

Participante	Abordagem	Tempo
P01	UML	64:01
P03	DSL	61:50
P05	UML	65:32
P06	DSL	51:10
P07	DSL	65:32
P08	DSL	84:32

Tabela 8 – Esforço por participante

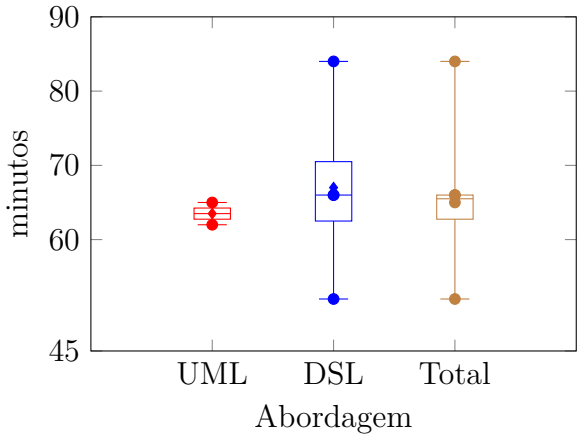


Figura 27 – Esforço dos participantes

posto para o sistema. Este tempo mostra-se inferior ao tempo de 1h06min40s referente ao esforço dos participantes para realizar a tarefa com a OpenMLPerf.

Na Tabela 8 é possível identificar o esforço de cada participante em realizar a modelagem. Percebe-se uma proximidade entre os tempos de modelagem com ambas as abordagens. A Figura 6.4 apresenta um resumo estatístico dos conjuntos de dados. A mediana da execução com a abordagem *UML* foi de 63.5 minutos, menor que a mediana com a abordagem utilizando a OpenMLPerf, que foi de 66 minutos. A abordagem com o perfil *UML* também teve um desvio padrão menor, de 2,12 minutos em relação ao desvio padrão da abordagem com OpenMLPerf, que foi de 13,11 minutos.

Como não houve uma amostra com tamanho satisfatório, não foi possível realizar a normalização e teste das hipóteses. Assim levamos em consideração para os resultados apenas os valores de estatística básica já apresentados.

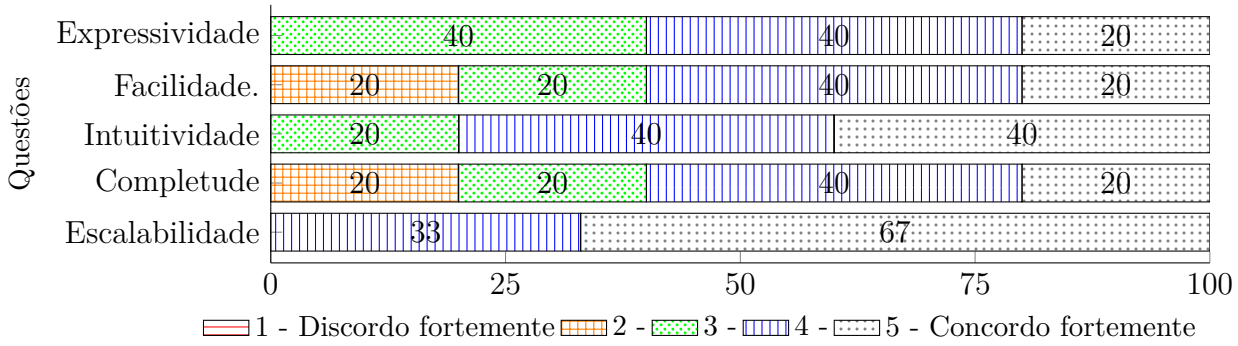
QP2. *Quão efetivo é modelar um teste de desempenho ao utilizar um perfil UML e a OpenMLPerf?* **QP3.** *Quão intuitivo/fácil é modelar testes de desempenho utilizando um perfil UML e a OpenMLPerf?*

Para responder estas duas questões, os participantes responderam a um survey após a realização da avaliação, onde foram questionados a respeito de conceitos relacionados a *expressividade, facilidade, intuitividade, completude, escalabilidade*. Para tal, podemos definir conceitualmente estes termos da seguinte forma: *Expressividade* - A capacidade da *DSL* em expressar de forma adequada requisitos funcionais e não funcionais relacionados ao sistema e a desempenho; *Facilidade* - A facilidade e clareza para modelar testes de desempenho utilizando a *DSL*; *Completude* - Referente a possibilidade de realizar a modelagem de cenários de teste com a *DSL* sem utilizar consulta a materiais de apoios; *Escalabilidade* - Se é possível aumentar a complexidade dos cenários modelados. Os participantes também responderam questões abertas com relação a modelagem com a OpenMLPerf e sugestões para a melhoria do experimento.

As questões foram estruturadas para serem respondidas utilizando a escala *Likert* para respostas de questionários. Após cada questão, o participante respondia escolhendo uma das afirmações: ***1 - Discordo fortemente, 2 - Discordo, 3 - Nem concordo ou discordo, 4 - Concordo, 5 - Concordo fortemente***

A Figura 28 mostra os dados obtidos com o survey. Com o compilado das respostas, podemos afirmar que: 60% (40% - Concordo, 20% - Concordo Fortemente) dos participantes consideram que a *DSL* possui expressividade, assim como, também 60% (40% - Concordo, 20% - Concordo Fortemente) dos participantes consideraram fácil utilizar a *DSL*. Em relação a intuitividade, 80% (40% - Concordo, 40% - Concordo Fortemente) dos respondentes consideram a OpenMLPerf intuitiva para realizar a modelagem. 60% (40% - Concordo, 20% - Concordo Fortemente) dos respondentes não sentiram necessidade de consulta a documentação para poder realizar a modelagem na *DSL*. Enquanto 100% dos participantes perceberam um bom nível de escalabilidade durante as modelagens (33% - Concordo, 67% - Concordo Fortemente).

Figura 28 – Resultados das questões sobre a OpenMLPerf



A Figura 29 apresenta o resultado da questão que perguntava se o participante recomendaria a utilização da OpenMLPerf a algum colega ou gerente. 80% dos participantes sugeriu que recomendariam utilização da *DSL*, enquanto 20% não a recomendariam.

Entre os benefícios de utilizar a abordagem com a OpenMLPerf, os participantes citaram, a facilidade, a intuitividade no uso e a praticidade ao inserir parâmetros, uma vez que não era necessário inserir *"taggedValues"* para cada parâmetro e propriedade de desempenho. Como desvantagem, os participantes citaram alguns problemas na usabilidade na hora de definir os parâmetros. Algumas sugestões de melhoria também foram mencionadas, como, alteração das imagens que representam a atividade inicial e a atividade final do diagrama de atividades que é a representação dos *scripts*. Os participantes também relataram problemas de performance ao executar a *DSL* no ambiente de desenvolvimento do Eclipse. Alguns participantes relataram ter achado um pouco desgastante a tarefa, sugerindo cenários e casos de usos mais simples e compactos.

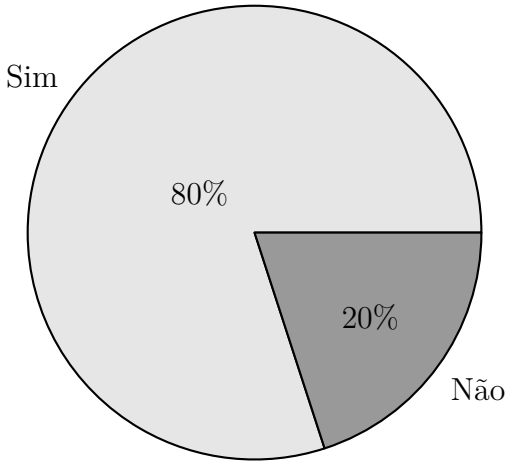


Figura 29 – Resultados sobre a recomendação do uso da *DSL*

6.4.1 Discussões

Nesta seção serão discutidos os resultados encontrados a respeito da OpenMLPerf e também em relação a execução da avaliação, assim como melhorias que possam ser aplicadas para melhorar uma nova execução da avaliação.

É observável uma proximidade entre os resultados obtidos com a modelagem utilizando as duas abordagens, pois não há grande significância entre a diferença nos esforços medidos para o perfil *UML*. Ao analisar friamente os dados obtidos com a avaliação, é possível perceber, como resultado, um menor esforço para realizar a modelagem com o perfil *UML* em relação a modelagem com a OpenMLPerf. Entretanto, devemos levar em consideração alguns fatores que possam ter influenciado nos resultados. Um exemplo são os problemas relatados como, queda de desempenho, travamentos e telas de carregamento muito demoradas no ambiente de desenvolvimento do Eclipse. Embora este problema, em uma avaliação empírica, possa ser mitigado com o uso de ambientes controlados e máquinas padronizadas, demonstra que para o uso individual ou em ambientes corporativos, que não possuam máquinas adequadas, o ambiente do Eclipse possa não ser a melhor

Abordagem	Grupo	Desvio Padrão
UML	1	1,767766953
OpenMLPerf	2	13,58096094

Tabela 9 – Desvio padrão para o esforço dos participantes

alternativa para o uso da linguagem desenvolvida. Outro fator contribuinte é a própria familiaridade dos participantes com o ambiente de desenvolvimento do perfil e a própria *UML*.

Como pode ser visto na Tabela 9, a não uniformidade entre os resultados do esforço com a abordagem com a OpenMLPerf também chama a atenção. O desvio padrão foi quase 13 vezes maior entre as duas abordagens (13,58096094 - *DSL, contra 1,767766953- perfil UML*. Como explicação para isso, podemos citar alguns fatores, como o participante com o maior tempo, que passou por problemas sérios de desempenho com o ambiente do Eclipse e não possuía experiencia com *DSLs* e com TDs. Enquanto o participante com o menor tempo, possuía um conhecimento regular, tanto em *DSLs*, como com TDs.

Outra questão a ser abordada em uma avaliação futura é a realização de múltiplas seções ou tarefas, para assim possibilitar o cruzamento dos resultados da execução das abordagens, permitindo que um grupo utilize ambas as abordagens e compare a modelagem de diversos pontos da linguagem, como cenário e *script*. Também pode ser realizada a verificação dos modelos criados, para identificação de erros de modelagem, tanto de sintaxe da escrita, como da modelagem das atividades. Também, como sugerido pelos participantes, simplificar e compactar os cenários e requisitos, para assim não perder o foco dos participantes e otimizar o tempo disponível para a realização das tarefas. Outra ação para mitigar os efeitos de curva de aprendizado é aumentar o tempo e as seções de treinamento em relação a ambas as abordagens e aos conceitos de teste de desempenho, uma vez que não deveremos ter novamente disponíveis especialistas nesta área.

Está previsto no prosseguimento do desenvolvimento da linguagem, a execução de uma nova avaliação, que será realizada futuramente com os cursos de computação e engenharia de software da UNIPAMPA

6.4.2 Lições do Capítulo

Neste capítulo foi apresentado uma avaliação da OpenMLPerf, onde foram observados fatores como, esforço, efetividade e facilidade. Assim foi possível identificar, através das respostas das questões de pesquisa, melhorias a serem implementadas, tanto na DSL, como na realização de um novo experimento. Em relação a DSL, aspectos podem receber melhorias, como em elementos gráficos que representam objetos do domínio de teste de desempenho, criação de parâmetros e questões de desempenho da LW que impactam na execução da linguagem, foram observados nos resultados obtidos. A execução do experimento nos permitiu verificar a importância do perfil do participante e o impacto desse

aspecto nos resultados obtidos. No próximo Capítulo serão apresentadas as considerações finais deste trabalho, além das lições aprendidas durante o desenvolvimento e avaliação da OpenMLPerf. Também serão discutidos os trabalhos futuros e direções da pesquisa.

7 CONSIDERAÇÕES FINAIS

Neste Capítulo, discutiremos as considerações finais deste trabalho. Ao final, serão discutidas os trabalhos futuros.

Testar sistemas complexos e ainda ter conhecimento das tecnologias necessárias para a realização deste teste, podem ser tarefas bastante custosas para uma equipe de software. Assim, um elemento da equipe que conheça e seja capaz de modelar o domínio e o sistema, independente da tecnologia utilizada em uma camada inferior na atividade de teste, é uma alternativa bastante atraente para solucionar questões relacionadas ao teste. Atividades do MDE, como o MBT, através do uso de DSLs, surgem como opções viáveis e práticas para o teste de sistemas em grande escala, tanto os já estabelecidos, como os novos, durante a fase de concepção.

Neste trabalho, apresentamos a implementação de uma DSL para modelagem de testes de desempenho para sistemas web, OpenMLPerf. Após apresentarmos conceitos relacionados ao domínio de modelagem de linguagens específicas de domínio, foram apresentadas os requisitos e decisões de design para o projeto da linguagem. Um mapeamento sistemático da literatura para a escolha da ferramenta mais adequada para a implementação também foi executado e detalhado neste trabalho. É apresentado uma avaliação empírica para avaliar o esforço do usuário ao realizar a modelagem com a DSL, onde foi possível comparar este esforço com o esforço para modelar teste de desempenho com um perfil UML.

7.1 Lições Aprendidas

Neste trabalho tivemos alguns desafios na escolha das tecnologias adequadas e no aprendizado sobre o seu correto uso. A execução do SMS auxiliou, entre outras coisas, a compreender o domínio de DSL e compreender as funcionalidades oferecidas pelas LW. Assim, limitar o número de ferramentas adequadas e decidir qual melhor se enquadraria nas necessidades da pesquisa tornou-se uma atividade menos custosa no ponto de vista do esforço de implementação. O que não significa que tornou mais fácil a implementação em si.

Trabalhar com tecnologias de ponta, ainda em desenvolvimento, sem o suporte adequado, ou muitas vezes inexistentes, é um dos ônus em tentar implementar sistemas que se propõem inovadores e relevantes. No caso deste trabalho, entender as diferenças entre modelos, metamodelos e até meta metamodelos foi uma tarefa bastante desafiadora. Diferenciar sintaxe abstrata e sintaxe semântica, diferenciar as *Eclass* do Ecore e classes *UML*, além de entender como cada elemento em um diagrama do Ecore teria influência em uma futura linguagem modelada no Sirius. Modelar todos os metamodelos da linguagem não foi uma atividade simples, já que os metamodelos da Canopus não poderiam ser reaproveitados, pois o MetaEdit+ usa o conceito de *GOPPRR (Graph, Object, Port, Property, Relationship e Role)* enquanto o Ecore tem seu conceito diferente, baseado em

classes e relacionamentos da UML, assim todos os metamodelos foram modelados do zero.

A implementação da linguagem com o framework Sirius não pode ser considerada uma tarefa difícil, uma vez que o propósito de LW é facilitar esta atividade, mas também não pode ser considerada trivial, principalmente por limitações da própria LW, que está em constante evolução. Então, grande parte da dificuldade encontrada resulta das limitações das curvas de aprendizado e falta de suporte em alguns recursos, como a criação dinâmica de modelos gerados por meios de recursos disponíveis em outros modelos (*i.e.,* modelo de métricas alocado em SUTs). Além de limitações comuns aos ambientes do Eclipse, como carregamentos demorados, erros e exceções.

A avaliação empírica realizada neste trabalho, apesar de suas limitações, não pode ser descartada, uma vez que, serve como base e pode ser considera um *trial* ou piloto para uma nova execução da avaliação. Permitindo assim, corrigir os erros encontrados e melhorar a generalidade das amostras, além de detectar limitações que fogem ao controle do desenvolvedor, como o desempenho e limitações de usabilidade do LW.

7.2 Trabalhos Futuros

A principal atividade a ser realizada futuramente, é a realização de uma nova avaliação empírica da OpenMLPerf. Buscando garantir um número maior e diversificado de participantes. Assim permitindo uma análise estatística mais robusta, precisa e satisfatória.

Para a linguagem em si, é esperado que futuramente possa ser integrada a uma notação textual para teste de desempenho, com a utilização do *framework* Xtext. Também devem ser implementados geradores de código, para permitir a exportação dos modelos como entrada para ferramentas de teste de desempenho.

REFERÊNCIAS

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. Citado na página 31.

ÅKESSON, A.; HEDIN, G. Jatte: a tunable tree editor for integrated DSLs. **Proceedings of the 2nd ACM SIGPLAN International Workshop on Comprehension of Complex Systems (CoCoS'17)**, p. 7–12, 2017. Citado na página 43.

AKHUNDOV, J. et al. Using timed automata to check space mission feasibility in the early design phases. In: **2016 IEEE Aerospace Conference**. [S.l.: s.n.], 2016. p. 1–9. Citado na página 44.

All4Tec. **Matelo**. 2019. Disponível em: <https://all4tec.com>. Citado na página 24.

ALVAREZ, C.; CASALLAS, R. MTC Flow: A Tool to Design, Develop and Deploy Model Transformation Chains. In: **Proceedings of the Workshop on ACadeMics Tooling with Eclipse**. New York, NY, USA: ACM, 2013. ((ACME'13)), p. 7:1—-7:9. ISBN 978-1-4503-2036-8. Citado na página 43.

ANTONELLI, H. L.; SILVA, E. A. N. da; FORTES, R. P. M. A Model-driven Development for Creating Accessible Web Menus. **Procedia Computer Science**, v. 67, p. 95–104, 2015. ISSN 1877-0509. Citado na página 43.

ARENDT, T.; TAENTZER, G.; WEBER, A. Quality assurance of textual models within eclipse using OCL and model transformations. In: **CEUR Workshop Proceedings**. Miami, FL, United states: [s.n.], 2013. v. 1092, p. 1–10. ISSN 16130073. Citado na página 43.

ARKIN, E.; TEKINERDOGAN, B. Domain specific language for deployment of parallel applications on parallel computing platforms. In: **ACM International Conference Proceeding Series (ICPS'14)**. Vienna, Austria: [s.n.], 2014. p. University of Vienna –. Citado na página 44.

Austrian Institute of Technology. **MoMut**. 2019. Disponível em: <https://momut.org/>. Citado na página 24.

BARIŠIĆ, A.; AMARAL, V.; GOULÃO, M. Usability driven DSL development with USE-ME. **Computer Languages, Systems and Structures**, v. 51, p. 1339–1351, 2017. ISSN 14778424. Citado na página 44.

BARIŠIĆ, A. et al. A Requirements Engineering Approach for Usability-driven DSL Development. In: **Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering**. New York, NY, USA: ACM, 2017. ((SLE'17)), p. 115–128. ISBN 978-1-4503-5525-4. Citado na página 43.

BARTMAN, B. et al. SrcQL: A syntax-aware query language for source code. In: **24th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER'17)**. [S.l.: s.n.], 2017. p. 467–471. Citado na página 44.

BERGENTI, F. An Introduction to the JADEL Programming Language. In: **2014 IEEE 26th International Conference on Tools with Artificial Intelligence (ICTAI'14)**. [S.l.: s.n.], 2014. p. 974–978. ISSN 1082-3409. Citado na página 43.

BERMUDEZ, R. et al. A tool to support the definition and enactment of model-driven migration processes. **Journal of Systems and Software**, v. 128, p. 106–129, 2017. ISSN 01641212. Citado na página 43.

BERNARDINO, M. Canopus: a domain-specific language for modeling performance testing. Pontifícia Universidade Católica do Rio Grande do Sul, 2016. Citado 4 vezes nas páginas 19, 29, 31 e 49.

BERNARDINO, M. et al. A systematic mapping study on model-based testing: Tools and models. **IET Software (2017)**, IET, 2017. Citado na página 37.

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. Citado na página 43.

BETTINI, L. Developing user interfaces with EMF parsley. In: **Proceedings of the 9th International Conference on Software Paradigm Trends (ICSOFT-PT'14)**. Vienna, Austria: [s.n.], 2014. p. 58–66. Citado na página 44.

BONNET, S. et al. Not (strictly) relying on SysML for MBSE: Language, tooling and development perspectives: The Arcadia/Capella rationale. **10th Annual International Systems Conference, (SysCon'16) - Proceedings**, 2016. Citado na página 43.

BUDINSKY, F. et al. **Eclipse modeling framework: a developer's guide**. [S.l.]: Addison-Wesley Professional, 2004. Citado na página 28.

CHALLENGER, M. et al. On the use of a domain-specific modeling language in the development of multiagent systems. **Engineering Applications of Artificial Intelligence**, v. 28, p. 111–141, 2014. ISSN 0952-1976. Citado na página 43.

CHLIPALA, A. et al. The end of history? Using a proof assistant to replace language design with library design. In: **Leibniz International Proceedings in Informatics, (LIPIcs'17)**. [S.l.: s.n.], 2017. v. 71. Citado na página 44.

COMBEMALE, B.; BARAIS, O.; WORTMANN, A. Language engineering with the GEMOC studio. **Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, (ICSAW'17): Side Track Proceedings**, p. 189–191, 2017. Citado na página 43.

CÓRDOBA-SÁNCHEZ, I.; LARA, J. de. Ann: A domain-specific language for the effective design and validation of Java annotations. **Computer Languages, Systems and Structures**, v. 45, p. 164–190, 2016. Citado na página 43.

De Faveri, C. D. et al. Towards security modeling of E-voting systems. **Proceedings - 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW'16)**, p. 145–154, 2017. Citado na página 44.

DEJANOVIĆ, I. et al. TextX: A Python tool for Domain-Specific Languages implementation. **Knowledge-Based Systems**, v. 115, p. 1–4, 2017. ISSN 09507051. Citado na página 43.

DEMIRKOL, S. et al. A DSL for the development of software agents working within a semantic web environment. **Computer Science and Information Systems**, v. 10, n. 4 SPEC.ISSUE, p. 1525–1556, 2013. Citado na página 43.

DWARAKANATH, A. et al. Accelerating Test Automation through a Domain Specific Language. **2017 IEEE International Conference on Software Testing, Verification and Validation (ICST'17)**, p. 460–467, 2017. Citado na página 43.

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>. Citado na página 29.

Eclipse Foundation. **Framework Sirius**. 2018. Available in: <https://www.eclipse.org/sirius//>. Citado na página 50.

EFFTINGE, S. et al. Xbase: Implementing Domain-specific Languages for Java. **SIGPLAN Not.**, ACM, New York, NY, USA, v. 48, n. 3, p. 112–121, 2012. ISSN 0362-1340. Citado 2 vezes nas páginas 44 e 50.

EL-FAR, I. K.; WHITTAKER, J. A. Model-based software testing. **Encyclopedia of Software Engineering**, Wiley Online Library, 2002. Citado na página 23.

ERDWEG, S. et al. The State of the Art in Language Workbenches. **Software Language Engineering**, v. 8225, p. 197–217, 2013. ISSN 03029743. Citado na página 44.

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. Citado na página 37.

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. Citado na página 37.

ERDWEG, S. et al. Evaluating and comparing language workbenches: Existing results and benchmarks for the future. In: **Computer Languages, Systems and Structures**. [S.l.: s.n.], 2015. v. 44, p. 24–47. ISSN 14778424. Citado na página 44.

FALKNER, K. et al. Modeling scenarios for the performance prediction of distributed real-time embedded systems. In: **Military Communications and Information Systems Conference (MilCIS'13) 2013**. [S.l.: s.n.], 2013. p. 1–6. Citado na página 43.

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. Citado na página 32.

FOWLER, M. **Domain-specific languages**. [S.l.]: Pearson Education, 2010. Citado 2 vezes nas páginas 19 e 25.

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. Citado 2 vezes nas páginas 19 e 23.

FUENTES-FERNÁNDEZ, L.; VALLECILLO-MORENO, A. An introduction to uml profiles. **UML and Model Engineering**, v. 2, 2004. Citado na página 26.

GARGANTINI, A.; VAVASSORI, P. CITLAB: A Laboratory for Combinatorial Interaction Testing. In: **Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation**. Washington, DC, USA: IEEE Computer Society, 2012. ((ICST'12)), p. 559–568. ISBN 978-0-7695-4670-4. Citado na página 43.

Gatling. **Gatling Stress Tool**. 2018. Disponível em: <http://gatling.io/>. Citado na página 33.

GIBBS, I.; DASCALU, S.; Harris Jr., F. C. A Separation-based UI Architecture with a DSL for Role Specialization. **Journal of Systems and Software.**, Elsevier Science Inc., New York, NY, USA, v. 101, n. C, p. 69–85, 2015. ISSN 0164-1212. Citado na página 43.

GRANCHELLI, G. et al. Towards recovering the software architecture of microservice-based systems. In: **Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, (ICSAW'17): Side Track Proceedings**. [S.l.: s.n.], 2017. p. 46–53. Citado na página 44.

GUELFI, N.; JAHIĆ, B.; RIES, B. TESMA: Requirements and design of a tool for educational programs. **Information (Switzerland)**, v. 8, n. 1, 2017. Citado na página 44.

HAITZER, T.; ZDUN, U. Semi-automated architectural abstraction specifications for supporting software evolution. **Science of Computer Programming**, v. 90, n. PART B, p. 135–160, 2014. ISSN 01676423. Citado na página 44.

HASAN, S. et al. A modeling framework to integrate exogenous tools for identifying critical components in power systems. **2017 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES'17)**, p. 1 – 6, 2017. Citado na página 43.

HÄSER, F.; FELDERER, M.; BREU, R. An integrated tool environment for experimentation in domain specific language engineering. In: **ACM International Conference Proceeding Series**. [S.l.: s.n.], 2016. ((ICPS'16), v. 01-03-June). Citado na página 43.

HEITKÖTTER, H. A Framework for Creating Domain-specific Process Modeling Languages. **Icsoft**, n. Mdd, p. 127–136, 2012. Citado na página 43.

HERRERA, A. S.-B. Enhancing xtext for general purpose languages. In: **CEUR Workshop Proceedings**. Valencia, Spain: [s.n.], 2014. v. 1321. ISSN 16130073. Citado na página 44.

Hewlett Packard. **Software HP LoadRunner**. 2018. Disponível em: <https://software.microfocus.com/pt-br/products/loadrunner-load-testing/free-trial>. Citado 2 vezes nas páginas 20 e 33.

HINKEL, G. et al. Using internal domain-specific languages to inherit tool support and modularity for model transformations. **Software and Systems Modeling**, p. 1–27, 2017. Citado na página 44.

HIYA, S. et al. clooca: Web based tool for domain specific modeling. In: **Demos/Posters/StudentResearch@ ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS'13)**. [S.l.: s.n.], 2013. p. 31–35. Citado na página 43.

HOISL, B.; SOBERNIG, S.; STREMBECK, M. Reusable and generic design decisions for developing UML-based domain-specific languages. **Information and Software Technology**, v. 92, p. 49–74, 2017. ISSN 09505849. Citado na página 43.

HOSEINDOOST, S. et al. A model-driven framework for developing multi-agent systems in emergency response environments. **Software and Systems Modeling**, p. 1–28, 2017. Citado na página 43.

HOYOS, J. R.; GARCIA-MOLINA, J.; BOTIA, J. A. A Domain-specific Language for Context Modeling in Context-aware Systems. **Journal of Systems and Software**, Elsevier Science Inc., New York, NY, USA, v. 86, n. 11, p. 2890–2905, 2013. ISSN 0164-1212. Citado na página 43.

HUANG, C. et al. Automated DSL Construction Based on Software Product Lines. In: **Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development**. Portugal: SCITEPRESS - Science and Technology Publications, Lda, 2015. ((MODELSWARD'15)), p. 247–254. ISBN 978-989-758-083-3. Citado na página 43.

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. Citado na página 68.

ILIASOV, A.; ROMANOVSKY, A. SafeCap domain language for reasoning about safety and capacity. **Proceedings - 2012 Workshop on Dependable Transportation Systems/Recent Advances in Software Dependability, (WDTS-RASD'12)**, p. 1–10, 2013. Citado na página 44.

JACOB, F. et al. Domain-specific languages for developing and deploying signature discovery workflows. **Computing in Science and Engineering**, v. 16, n. 1, p. 52–64, 2014. ISSN 15219615. Citado na página 44.

JAFER, S.; CHHAYA, B.; DURAK, U. Graphical specification of flight scenarios with aviation scenario definition language (ASDL). In: **AIAA Modeling and Simulation Technologies Conference, 2017 (AIAA SciTech'17)**. [S.l.: s.n.], 2017. Citado na página 43.

JAGER, S. et al. Creation of domain-specific languages for executable system models with the Eclipse Modeling Project. In: **10th Annual International Systems Conference, (SysCon'16) - Proceedings**. [S.l.: s.n.], 2016. Citado na página 44.

JÉZÉQUEL, J.-M. et al. Mashup of Metalanguages and Its Implementation in the Kermeta Language Workbench. **Software and Systems Modeling**, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 14, n. 2, p. 905–920, 2015. ISSN 1619-1366. Citado na página 43.

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. Citado na página 26.

KELLY, S.; TOLVANEN, J.-P. **Domain-Specific Modeling: Enabling Full Code Generation**. New York, NY, USA: John Wiley & Sons, 2007. ISBN 0470036664. Citado na página 25.

KENT, S. Model driven engineering. In: SPRINGER. **International Conference on Integrated Formal Methods**. [S.l.], 2002. p. 286–298. Citado na página 23.

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. Citado na página 26.

KORENKOV, Y.; LOGINOV, I.; LAZDIN, A. PEG-based language workbench. In: **Conference of Open Innovation Association, (FRUCT'15)**. Yaroslavl, Russia: [s.n.], 2015. v. 2015-June, n. June, p. 75–81. ISSN 23057254. Citado na página 43.

KOWALSKI, M.; MAGOTT, J. Time coordination of heterogeneous distance protections using a domain specific language. **E-Informatica Software Engineering Journal**, v. 6, n. 1, p. 7–26, 2012. Citado na página 44.

KRASTS, O.; KLEINS, A.; TEILANS, A. Domain specific language for securities settlement systems. In: **Digital Information Processing and Communications (ICDIPC'12), Second International Conference on**. [S.l.: s.n.], 2012. p. 80–83. Citado na página 44.

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. Citado na página 19.

Le Goaer, O.; WALTHAM, S. Yet Another DSL for Cross-platforms Mobile Development. In: **Proceedings of the First Workshop on the Globalization of Domain Specific Languages**. New York, NY, USA: ACM, 2013. ((GlobalDSL'13)), p. 28–33. ISBN 978-1-4503-2043-6. Citado na página 44.

LEMAZURIER, L.; CHAPURLAT, V.; GROSSETÊTE, A. An MBSE Approach to Pass from Requirements to Functional Architecture. **IFAC-PapersOnLine**, Elsevier B.V., v. 50, n. 1, p. 7260–7265, 2017. ISSN 24058963. Citado na página 43.

MA, T.; SALLAI, J. MiW: A domain specific modeling environment for complex molecular systems. **Procedia Computer Science**, v. 108, p. 1232–1241, 2017. ISSN 18770509. Citado na página 43.

MARAND, E. A.; MARAND, E. A.; CHALLENGER, M. DSML4CP: A Domain-specific Modeling Language for Concurrent Programming. **Computer Languages, Systems and Structures**, v. 44, p. 319–341, 2015. ISSN 14778424. Citado na página 44.

MARO, S. et al. On Integrating Graphical and Textual Editors for a UML Profile Based Domain Specific Language: An Industrial Experience. In: **Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering**. New York, NY, USA: ACM, 2015. ((SLE'15)), p. 1–12. ISBN 978-1-4503-3686-4. Citado na página 43.

MARÓTI, M. et al. Next generation (Meta)modeling: Web- and cloud-based collaborative tool infrastructure. **CEUR Workshop Proceedings**, v. 1237, p. 41–60, 2014. ISSN 16130073. Citado na página 43.

MAVROPOULOS, O. et al. ASTo: A tool for security analysis of IoT systems. In: **Proceedings - 2017 15th IEEE/ACIS International Conference on Software Engineering Research, Management and Applications, (SERA'17)**. [S.l.: s.n.], 2017. p. 395–400. Citado na página 43.

MAYR-DORN, C.; LAABER, C. A Domain-Specific Language for Coordinating Collaboration. **2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA'17)**, p. 57–60, 2017. Citado na página 43.

MEIER, J. et al. **Performance testing guidance for web applications: patterns & practices**. [S.l.]: Microsoft press, 2007. Citado na página 24.

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. Citado na página 25.

MetaCase. **MetaEdit+ Modeler**. 2018. Disponível em: <https://www.metacase.com/mep/>. Citado 2 vezes nas páginas 20 e 29.

MOHAMAD, R. P.; KOLOVOS, D. S.; PAIGE, R. F. Resource requirement analysis for web applications running in a virtualised environment. **Proceedings of the International Conference on Cloud Computing Technology and Science, (CLOUDCOM'15)**, v. 2015-Febru, n. February, p. 632–637, 2015. Citado na página 43.

MOLINA, A. I. et al. Metamodel-driven Definition of a Visual Modeling Language for Specifying Interactive Groupware Applications: An Empirical Study. **Journal of Systems and Software**, Elsevier Science Inc., New York, NY, USA, v. 86, n. 7, p. 1772–1789, 2013. ISSN 0164-1212. Citado na página 43.

MONTENEGRO-MARIN, C. E. et al. Domain specific language for the generation of learning management systems modules. **Journal of Web Engineering**, v. 11, n. 1, p. 23, 2012. Citado na página 44.

MONTHE, V. M. et al. RsaML: A domain specific modeling language for describing robotic software architectures with integration of real-time properties. In: **CEUR Workshop Proceedings**. [S.l.: s.n.], 2016. v. 1697. Citado na página 44.

MONTRIEUX, L.; YU, Y.; WERMELINGER, M. Developing a domain-specific plug-in for a modelling platform: The good, the bad, the ugly. In: **2013 3rd International Workshop on developing Tools as Plug-ins (TOPI'13)**. [S.l.: s.n.], 2013. p. 1–6. ISSN 2327-0748. Citado na página 44.

NAGELE, T.; HOOMAN, J. Rapid Construction of Co-Simulations of Cyber-Physical Systems in HLA Using a DSL. **2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA'17)**, p. 247–251, 2017. Citado na página 43.

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. Citado 2 vezes nas páginas 31 e 43.

NAUJOKAT, S. et al. Cinco: A simplicity-driven approach to full generation of domain-specific graphical modeling tools. **International Journal on Software Tools for Technology Transfer**, Springer, p. 1–28, 2017. Citado na página 43.

NAZIR, A. et al. A high-level domain-specific language for SIEM (design, development and formal verification). **Cluster Computing**, v. 20, n. 3, p. 2423–2437, 2017. Citado na página 43.

NEUBAUER, P. et al. Automated generation of consistency-achieving model editors. In: IEEE. **24th International Conference on Software Analysis, Evolution and Reengineering (SANER'17), 2017 IEEE**. [S.l.], 2017. p. 127–137. Citado na página 43.

NORDMANN, A.; WREDE, S.; STEIL, J. Modeling of movement control architectures based on motion primitives using domain-specific languages. In: **Proceedings - IEEE International Conference on Robotics and Automation (ICRA'15)**. Seattle, WA, United states: [s.n.], 2015. v. 2015-June, n. June, p. 5032–5039. ISSN 10504729. Citado na página 43.

OLIVEIRA, B.; BELO, O. On the specification of extract, transform, and load patterns behavior: A domain-specific language approach. **Expert Systems**, v. 34, n. 1, 2017. Citado na página 43.

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. Citado na página 31.

PESCADOR, A. et al. Pattern-based development of Domain-Specific Modelling Languages. In: **2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems, (MODELS'15) - Proceedings**. Ottawa, ON, Canada: [s.n.], 2015. p. 166–175. Citado na página 43.

PESCADOR, A.; LARA, J. D. DSL-Maps: From Requirements to Design of Domain-Specific Languages. **31st IEEE/ACM International Conference on Automated Software Engineering (ASE'16)**, p. 438–443, 2016. Citado na página 44.

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. Citado na página 35.

POMANTE, L.; CANDIA, S.; INCERTO, E. A Model-Driven approach for the development of an IDE for Spacecraft on-board software. In: **2015 IEEE Aerospace Conference**. [S.l.: s.n.], 2015. p. 1–17. ISSN 1095-323X. Citado na página 43.

RATIU, D.; ULRICH, A. Increasing usability of spin-based C code verification using a harness definition language: Leveraging model-driven code checking to practitioners. In: **SPIN 2017 - Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software (SPIN'17)**. [S.l.: s.n.], 2017. p. 60–69. Citado na página 43.

RATIU, D.; VOELTER, M. Automated Testing of DSL Implementations - Experiences from Building mbeddr. In: **2016 IEEE/ACM 11th International Workshop in Automation of Software Test (AST'16)**. [S.l.: s.n.], 2016. p. 15–21. Citado na página 43.

REIN, P.; HIRSCHFELD, R.; TAEUMEL, M. Gramada: Immediacy in programming language development. In: **Onward! 2016 - Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (SPLASH'16)**. [S.l.: s.n.], 2016. p. 165–179. Citado na página 44.

RIBEIRO, A.; Da Silva, A. R. XIS-Mobile: A DSL for mobile applications. In: **Proceedings of the ACM Symposium on Applied Computing**. [S.l.: s.n.], 2014. ((SIGAPP'14)), p. 1316–1323. Citado na página 44.

RIBEIRO, A.; De Sousa, L.; Da Silva, A. R. Comparative analysis of workbenches to support DSMLs: Discussion with non-trivial model-driven development needs. In: **Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD'16)**. [S.l.: s.n.], 2016. p. 323–330. Citado na página 44.

RIBEIRO, A.; SILVA, A. R. da. RSLingo4Privacy Studio - A Tool to Improve the Specification and Analysis of Privacy Policies. **Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS'17)**, n. September, p. 52–63, 2017. Citado na página 43.

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. Citado 2 vezes nas páginas 19 e 31.

ROCHA, H. et al. DCL 2.0: modular and reusable specification of architectural constraints. **Journal of the Brazilian Computer Society**, v. 23, n. 1, 2017. Citado na página 44.

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. Citado na página 65.

ROSE, L. M.; KOLOVOS, D. S.; PAIGE, R. F. EuGENia Live: A Flexible Graphical Modelling Tool. In: **Proceedings of the 2012 Extreme Modeling Workshop (XM'2012)**. New York, NY, USA: ACM, 2012. (XM'12), p. 15–20. ISBN 978-1-4503-1804-4. Citado na página 44.

ROZEN, R. van et al. Toward live domain-specific languages: From text differencing to adapting models at run time. **Software & Systems Modeling**, Springer Berlin Heidelberg, 2017. ISSN 1619-1374. Citado na página 44.

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. Citado 2 vezes nas páginas 31 e 32.

SANDOBALIN, J.; INSFRAN, E.; ABRAHAO, S. An Infrastructure Modelling Tool for Cloud Provisioning. **Proceedings - 2017 IEEE 14th International Conference on Services Computing (SCC'17)**, p. 354–361, 2017. Citado na página 43.

SANTOS, A. L.; GOMES, E. Xdiagram: A declarative textual DSL for describing diagram editors (Tool Demo -). In: **Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering, co-located with SPLASH 2016 (SLE'16)**. [S.l.: s.n.], 2016. ((SLE'16)), p. 253–257. Citado na página 44.

SAVIĆ, D. et al. Preliminary experience using JetBrains MPS to implement a requirements specification language. **Proceedings - 2014 9th International Conference on the Quality of Information and Communications Technology, (QUATIC'14)**, n. 1, p. 134–137, 2014. Citado na página 43.

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. Citado na página 24.

SMITS, J.; VISSER, E. FlowSpec: Declarative Dataflow Analysis Specification. In: **Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering**. New York, NY, USA: ACM, 2017. ((SLE'17)), p. 221–231. ISBN 978-1-4503-5525-4. Citado na página 44.

SOUSA, L. M. D.; SILVA, A. R. D. A domain specific language for spatial simulation scenarios. **GeoInformatica**, Springer, v. 20, n. 1, p. 117–149, 2016. Citado na página 43.

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. Citado na página 33.

STEINBERG, D. et al. **EMF: eclipse modeling framework**. [S.l.]: Pearson Education, 2008. Citado 2 vezes nas páginas 27 e 28.

STOCKER, K. b.; WASHIZAKI, H.; FUKAZAWA, Y. Closing the Gap between Unit Test Code and Documentation. **Proceedings - 10th IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW'17)**, p. 304–308, 2017. Citado na página 43.

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. Citado 2 vezes nas páginas 32 e 33.

SUTII, A. M.; BRAND, M. van den; VERHOEFF, T. Exploration of modularity and reusability of domain-specific languages: An expression DSL in MetaMod. **Computer Languages, Systems and Structures**, 2017. ISSN 14778424. Citado na página 44.

SZABO, T. et al. An extensible framework for variable-precision data-flow analyses in MPS. In: **Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering**. Singapore, Singapore: [s.n.], 2016. ((ASE'16)), p. 870–875. Citado na página 43.

TAIRAS, R.; CABOT, J. Corpus-based Analysis of Domain-specific Languages. **Software and Systems Modeling**, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 14, n. 2, p. 889–904, 2015. ISSN 1619-1366. Citado na página 44.

TANG, W. Meta object facility. In: _____. **Encyclopedia of Database Systems**. Boston, MA: Springer US, 2009. p. 1722–1723. ISBN 978-0-387-39940-9. Citado na página 27.

TARIQ, M. U.; FLORENCE, J.; WOLF, M. Design specification of cyber-physical systems: Towards a domain-specific modeling language based on simulink, eclipse modeling framework, and giotto. In: **CEUR Workshop Proceedings**. [S.l.: s.n.], 2014. v. 1250, p. 6–15. Citado na página 44.

Test Optimal. **Test Optimal**. 2019. Disponível em: <http://mbt.testoptimal.com/index.html>. Citado na página 24.

TIKHONOVA, U. Reusable specification templates for defining dynamic semantics of DSLs. **Software and Systems Modeling**, p. 1–30, 2017. Citado na página 44.

UHNÁK, P.; PERGL, R. The OpenPonk Modeling Platform. In: **Proceedings of the 11th Edition of the International Workshop on Smalltalk Technologies**. New York, NY, USA: ACM, 2016. ((IWST'16)), p. 14:1—-14:11. ISBN 978-1-4503-4524-8. Citado na página 44.

UTTING, M.; LEGEARD, B. **Practical model-based testing: a tools approach**. [S.l.]: Elsevier, 2010. Citado na página 23.

VIANA, M. et al. F3T: From features to frameworks tool. In: **Proceedings - 2013 27th Brazilian Symposium on Software Engineering, (SBES'13)**. Brasilia, DF, Brazil: [s.n.], 2013. p. 89–98. Citado na página 44.

VIANA, M. C.; PENTEADO, R. A. D.; PRADO, A. F. do. Domain-Specific Modeling Languages to improve framework instantiation. **Journal of Systems and Software**, v. 86, n. 12, p. 3123–3139, 2013. ISSN 0164-1212. Citado na página 44.

VIEIRA, M. A.; CARVALHO, S. T. Model-driven Engineering in the Development of Ubiquitous Applications: Technologies, Tools and Languages. In: **Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web**. New York, NY, USA: ACM, 2017. ((WebMedia'17)), p. 29–32. ISBN 978-1-4503-5096-9. Citado na página 43.

VINOGRADOV, S.; OZHIGIN, A.; RATIU, D. Modern model-based development approach for embedded systems practical experience. In: **1st IEEE International Symposium on Systems Engineering, (ISSE'15) - Proceedings**. Rome, Italy: [s.n.], 2015. p. 56–59. Citado na página 43.

VISIC, N. et al. A domain-specific language for modeling method definition: From requirements to grammar. In: **Proceedings - International Conference on Research Challenges in Information Science**. Athens, Greece: [s.n.], 2015. ((RCIS'15), June), p. 286–297. ISSN 21511349. Citado na página 43.

VISSERS, Y. et al. Maintenance of specification models in industry using edapt. In: IEEE. **Specification and Design Languages (FDL), 2017 Forum on**. [S.l.], 2017. p. 1–6. Citado na página 43.

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. Citado 2 vezes nas páginas 28 e 44.

VOELTER, M. et al. Lessons learned from developing mbeddr: a case study in language engineering with MPS. **Software and Systems Modeling**, p. 1–46, 2017. Citado na página 43.

VOELTER, M. et al. Mbeddr: An extensible c-based programming language and IDE for embedded systems. In: **Proceedings of the 2012 ACM Conference on Systems, Programming, and Applications: Software for Humanity (SPLASH'12)**. Tucson, AZ, United states: [s.n.], 2012. p. 121–140. Citado na página 43.

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. Citado na página 26.

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. Citado na página 43.

WALTER, T.; PARREIRAS, F. S.; STAAB, S. An ontology-based framework for domain-specific modeling. **Software & Systems Modeling**, v. 13, n. 1, p. 83–108, 2014. ISSN 1619-1374. Citado na página 43.

WIGAND, D. L. et al. Modularization of domain-specific languages for extensible component-based robotic systems. **Proceedings - 2017 1st IEEE International Conference on Robotic Computing, (IRC'17)**, p. 164–171, 2017. Citado na página 43.

ZARRIN, B.; BAUMEISTER, H. Design of a Domain-Specific Language for Material Flow Analysis Using Microsoft DSL Tools: An Experience Paper. In: **Proceedings of the 14th Workshop on Domain-Specific Modeling**. New York, NY, USA: ACM, 2014. ((DSM'14)), p. 23–28. ISBN 978-1-4503-2156-3. Citado na página 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. Citado na página 19.

ZHOU, N. et al. Model-based development of knowledge-driven self-reconfigurable machine control systems. **IEEE Access**, IEEE, v. 5, p. 19909–19919, 2017. Citado na página 43.

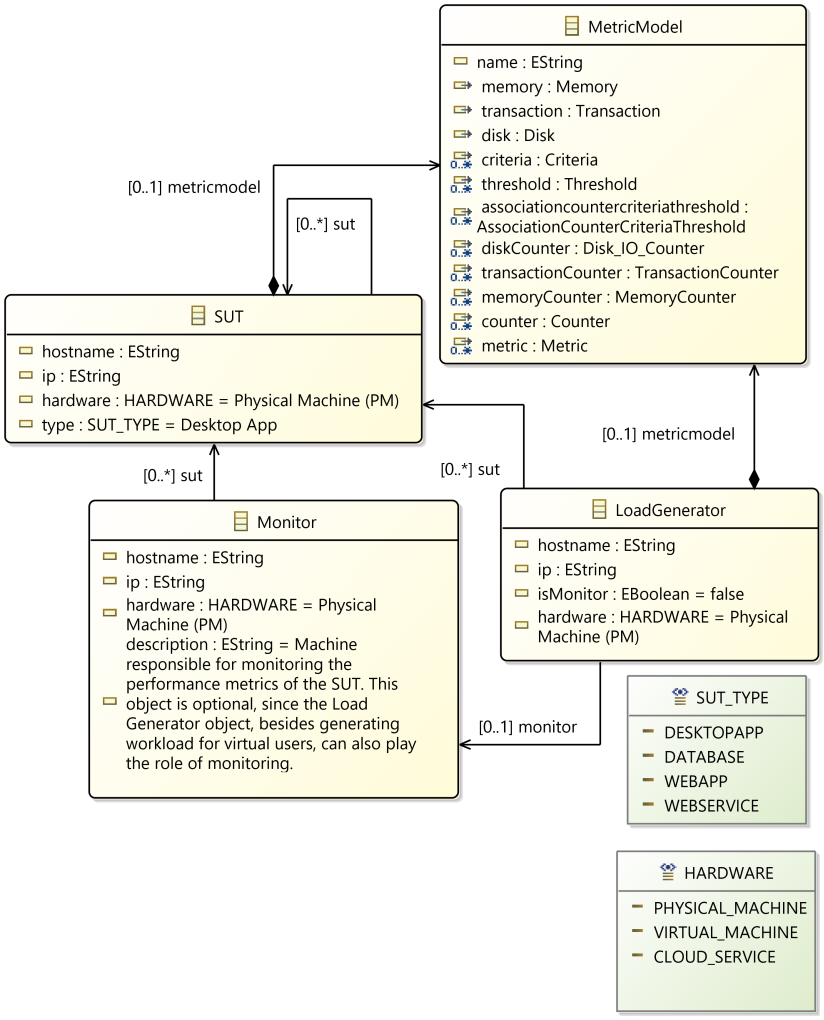
ZHU, Z. et al. Cognitive behaviors modeling using uml profile: Design and experience. **IEEE Access**, IEEE, v. 5, p. 21694–21708, 2017. Citado na página 43.

ZIKRA, I. LNBIP 134 - Implementing the Unifying Meta-model for Enterprise Modeling and Model-Driven Development: An Experience Report. p. 172–187, 2012. Citado na página 43.

Apêndices

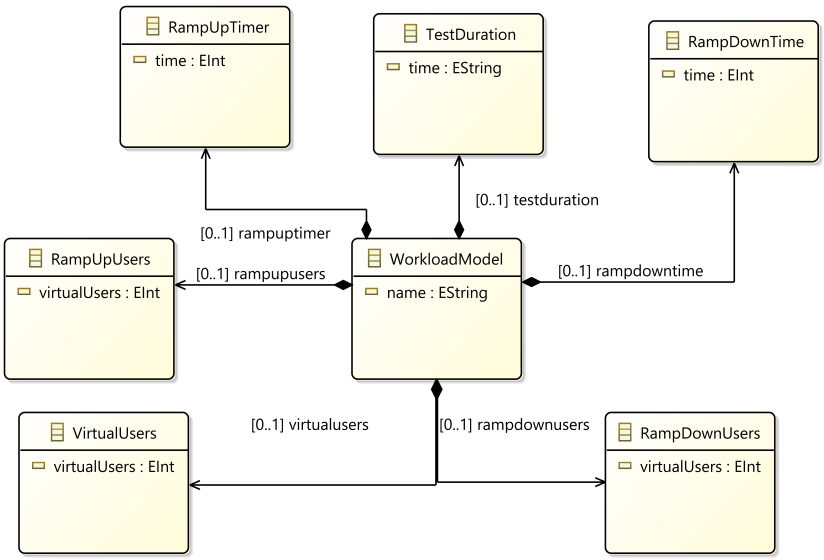
APÊNDICE A – APÊNDICE A

Figura 30 – Diagrama de classes exibindo o metamodelo de monitoramento da linguagem



Fonte: Autor

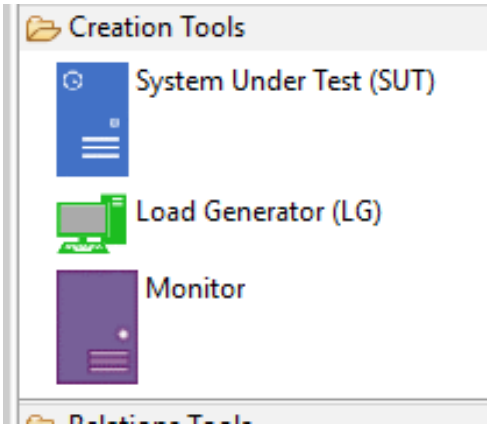
Figura 31 – Diagrama de classes exibindo o metamodelo da carga de trabalho da linguagem



Fonte: Autor

APÊNDICE B – APÊNDICE B

Figura 32 – Diagrama de modelagem dos *scripts* da linguagem



Fonte: Autor

APÊNDICE C – APÊNDICE C

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.

ÍNDICE