

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ONTOLOGIA PARA TESTE DE DESEMPENHO DE SOFTWARE

ARTUR LUIZ SILVA
DA CUNHA FREITAS

Dissertação apresentada como requisito parcial
à obtenção do grau de Mestre em Ciência da
Computação na Pontifícia Universidade Católica
do Rio Grande do Sul.

Orientadora: Prof^a. Dr^a. Renata Vieira

**Porto Alegre
2013**

Dados Internacionais de Catalogação na Publicação (CIP)

F866o Freitas, Artur Luiz Silva da Cunha
Ontologia para teste de desempenho de software / Artur Luiz Silva da
Cunha Freitas. – Porto Alegre, 2013.
151 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof^a. Dr^a. Renata Vieira.

1. Informática. 2. Ontologia. 3. Engenharia de Software. 4. Software –
Análise de Desempenho. I. Vieira, Renata. II. Título.

CDD 005.1

**Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS**



TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Ontologia para Teste de Desempenho de Software" apresentada por Artur Luiz Silva da Cunha Freitas como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Inteligência Computacional, aprovada em 08/03/2013 pela Comissão Examinadora:

Prof. Dra. Renata Vieira -
Orientadora

PPGCC/PUCRS

Prof. Dr. Rafael Heitor Bordini -

PPGCC/PUCRS

Prof. Dr. Flávio Moreira de Oliveira -

FACIN/PUCRS

Homologada em 30/04/2013, conforme Ata No. 007 pela Comissão Coordenadora.

Prof. Dr. Paulo Henrique Lemelle Fernandes
Coordenador.

PUCRS

Campus Central

Av. Ipiranga, 6681 - P32 - sala 507 - CEP: 90619-900

Fone: (51) 3320-3611 - Fax (51) 3320-3621

E-mail: ppgcc@pucrs.br

www.pucrs.br/facin/pos

AGRADECIMENTOS

Agradeço aos meus familiares e amigos pelo apoio e bons momentos que passamos juntos. Destaco minha gratidão em especial à professora Renata Vieira pela orientação e ao professor Flávio de Oliveira pela oportunidade de participar do projeto de pesquisa. Agradeço à PUCRS pelo programa de pós-graduação em ciência da computação e à Dell pela bolsa de estudos e troca de experiências.

“Your present circumstances don't determine where you can go; they merely determine where you start.” (Nido Qubein)

ONTOLOGIA PARA TESTE DE DESEMPENHO DE SOFTWARE

RESUMO

O teste de software é uma técnica utilizada para fornecer informações sobre a qualidade de um software operando em um contexto específico. O tipo de teste responsável por avaliar o desempenho e a eficiência de um software em um cenário de uso é conhecido como teste de desempenho. A elaboração de um teste de desempenho é uma tarefa que exige testadores com conhecimentos especializados referentes às ferramentas, atividades e métricas do domínio. Para representar tais conhecimentos, este trabalho propõe uma ontologia sobre o domínio de teste de desempenho. Ontologia é uma técnica de representação do conhecimento considerada o estado da arte dentro da Inteligência Artificial. Além disso, uma das contribuições desta pesquisa é mapear, com base na análise de trabalhos relacionados, o que se sabe sobre a utilização de ontologias no teste de software. Por fim, a ontologia proposta é avaliada por especialistas do domínio, comparada com ontologias relacionadas e explorada em aplicações que têm por objetivo auxiliar os testadores quanto ao planejamento e à elaboração dos testes de desempenho.

Palavras Chave: ontologia, teste de software, teste de desempenho.

ONTOLOGY FOR SOFTWARE PERFORMANCE TESTING

ABSTRACT

Software testing is a technique used to provide information about the quality of a software operating in a specific context. The test responsible for assessing the performance and efficiency of a software in a usage scenario is known as performance testing. The development of a performance test is a task that requires testers with expertise related to the tools, activities and metrics of the domain. To represent this knowledge, this work proposes an ontology on the domain of performance testing. Ontology is a knowledge representation technique considered state of the art within the Artificial Intelligence. Moreover, one contribution of this research is to identify, based on the analysis of related works, what is known about the use of ontologies in software testing. Finally, the proposed ontology is evaluated by domain experts, compared with related ontologies and explored in applications designed to help testers regarding the planning and elaboration of performance tests.

Keywords: ontology, software testing, performance testing.

LISTA DE FIGURAS

Figura 1: Ciclo de vida de construção de uma ontologia (Fonte [29]).	32
Figura 2: Interface gráfica do Protégé versão 4.2.	34
Figura 3: Modelo do domínio de desempenho (Fonte [29]).	40
Figura 4: Regra em lógica de primeira ordem para definir <i>HardWork</i> (Fonte [29]).	41
Figura 5: Regra em lógica de primeira ordem para definir <i>HighPriorityTask</i> (Fonte [29]).	41
Figura 6: Regra em lógica de primeira ordem para definir <i>EfficientExecutionWork</i> (Fonte [29]).	41
Figura 7: Definição de uma Action-Rule para garantia da Qualidade de Serviço (Fonte [29]).	42
Figura 8: Abordagem dinâmica de aperfeiçoamento do desempenho (Fonte [11]).	44
Figura 9: Exemplo de um Grafo de Interdependência de Softgoals (Fonte [56]).	45
Figura 10: Estrutura da OntoTest (Fonte [19]).	49
Figura 11: Abordagem baseada em ontologias para geração de casos de teste unitário (Fonte [71]).	50
Figura 12: Robustness Testing Framework (Fonte [42]).	52
Figura 13: Visão geral do framework para geração de teste baseado em ontologias (Fonte [70]).	53
Figura 14: Abordagem baseada em ontologias para geração de testes para <i>web services</i> (Fonte [74]).	56
Figura 15: Diagrama de casos de uso descrevendo o escopo de aplicações baseadas na ontologia proposta.	58
Figura 16: Hierarquia dos conceitos do domínio (imagem gerada com o plugin OWLViz do Protégé).	62
Figura 17: Restrições sobre o conceito <i>PerformanceTest</i> .	63
Figura 18: Conceitos que especializam <i>PTGoal</i> e restrições sobre <i>NormalWorkloadPerformanceEvaluation</i> .	64
Figura 19: Hierarquia das atividades dos testes de desempenho adotada na ontologia.	65
Figura 20: Restrições do conceito <i>PTCaseDevelopment</i> .	66
Figura 21: Restrições dos conceitos <i>PTCaseSpecificationElaboration</i> e <i>PTReportElaboration</i> .	66
Figura 22: Restrições dos conceitos <i>PTCaseExecution</i> e <i>PerformanceMetricMonitoring</i> .	67
Figura 23: Restrições dos conceitos <i>PerformanceToolsDefinition</i> e <i>WorkloadDefinition</i> .	67
Figura 24: Hierarquia de conceitos dos entregáveis dos testes de desempenho.	68
Figura 25: Hierarquia e instâncias das subclasses de <i>PerformanceMetric</i> .	70
Figura 26: Hierarquia de conceitos e instâncias das ferramentas de teste de desempenho.	72
Figura 27: Restrições sobre os conceitos utilizados pelo <i>reasoner</i> para classificar instâncias de <i>PTTool</i> .	73
Figura 28: Hierarquia dos conceitos do sistema sob teste e padrão <i>ValuePartition</i> .	74
Figura 29: Propriedades representadas na ontologia.	75
Figura 30: Diagrama de classe resumindo os principais conceitos e relacionamentos da ontologia.	78
Figura 31: Qualidade das ontologias (Fonte [34]).	80
Figura 32: Legenda utilizada para definir a expressividade em Description Logic adotada pelo Protégé.	84
Figura 33: Personalização das abas em exibição na interface do Protégé.	94
Figura 34: Exemplo de grafo de dependência de atividades gerado pelo plugin <i>ActivityDependencies</i> .	95

Figura 35: Demonstração do plugin <i>RestrictionsWidget</i> (painel com borda verde) sendo utilizado pelo Protégé.	96
Figura 36: Criação de uma nova instância do conceito <i>PerformanceTest</i>	97
Figura 37: Edição de informações relacionadas a uma instância do conceito <i>PerformanceTest</i>	97
Figura 38: Exemplo de consultas em SQWRL sobre a ontologia de teste de desempenho.	100
Figura 39: Resultados obtidos pela aplicação do <i>reasoner</i> Pellet sobre as instâncias da ontologia.	101
Figura 40: Tela inicial do WebProtégé hospedando a ontologia de teste de desempenho.	102
Figura 41: WebProtégé exibindo as propriedades do indivíduo <i>LoadRunner</i> (pertencente ao conceito <i>PTTool</i>).	103
Figura 42: WebProtégé exibindo as informações do conceito <i>PerformanceBottleneckIdentification</i>	103
Figura 43: Interface da aplicação para planejamento e gerência de testes de desempenho.	104
Figura 44: Código fonte para criar uma instância de um conceito na ontologia utilizando a OWL API.	106
Figura 45: Plano de teste para a instância de <i>PerformanceTest</i> denominada “Skills Performance Test”.	107
Figura 46: Exemplo de uso da iText para criar um PDF.	108
Figura 47: Instruções gerais para responder o questionário.	123
Figura 48: Informações sobre a experiência e conhecimentos do participante em teste de desempenho.	124
Figura 49: Perguntas sobre os objetivos dos testes de desempenho respondidas pelos especialistas.	125
Figura 50: Nível de concordância dos especialistas sobre as subclasses do conceito <i>PTGoal</i>	126
Figura 51: Nível de concordância dos especialistas sobre as subclasses do conceito <i>PTArtifact</i>	127
Figura 52: Nível de concordância dos especialistas sobre as subclasses do conceito <i>PTActivity</i>	128
Figura 53: Nível de concordância dos especialistas em relação às categorias do tipo <i>PTActivity</i>	128
Figura 54: Opção para os especialistas registrarem comentários adicionais sobre o questionário.	129
Figura 55: Mensagem de agradecimento exibida para os participantes que concluíram o questionário.	129
Figura 56: Objetivos dos testes de desempenho mais frequentemente citados pelos especialistas.	144
Figura 57: Artefatos dos testes de desempenho mais frequentemente citados pelos especialistas.	145
Figura 58: Ferramentas de teste de desempenho identificadas pelos especialistas.	147
Figura 59: Atividades dos testes de desempenho identificadas pelos especialistas.	149
Figura 60: Métricas de desempenho identificadas com maior frequência pelos especialistas.	151

LISTA DE TABELAS

Tabela 1: Principais instâncias de conceitos que especializam o conceito <i>PerformanceMetric</i>	71
Tabela 2: Comparação entre os conceitos da ontologia proposta com a OntoTest [19] e a SwTO [13].	81
Tabela 3: Métricas extraídas do Protégé para comparação das ontologias.	83
Tabela 4: Formação acadêmica dos especialistas entrevistados.	86
Tabela 5: Comparação entre as respostas dos especialistas e os elementos da ontologia.	90
Tabela 6: Avaliação geral dos especialistas sobre os conceitos da ontologia.	91
Tabela 7: Restrições dos conceitos definidos utilizados para aplicar o <i>reasoner</i> sobre instâncias da ontologia.	101
Tabela 8: Comparação resumida das contribuições dos trabalhos relacionados.	110
Tabela 9: Comparação das contribuições dos trabalhos relacionados (parte 1).	112
Tabela 10: Comparação das contribuições dos trabalhos relacionados (parte 2).	113
Tabela 11: Ocupações dos especialistas relacionadas ao teste de desempenho.	131
Tabela 12: Como o conhecimento em teste de desempenho foi adquirido pelos especialistas.	131
Tabela 13: Objetivos dos testes de desempenho levantados pelos especialistas.	132
Tabela 14: Ferramentas para teste de desempenho citadas pelos especialistas.	133
Tabela 15: Atividades dos testes de desempenho segundo os especialistas entrevistados.	134
Tabela 16: Como agrupar as atividades dos testes de desempenho.	135
Tabela 17: Entregáveis que podem resultar de um teste de desempenho.	136
Tabela 18: Métricas de desempenho que podem ser monitoradas em um teste.	137
Tabela 19: Como agrupar as métricas de desempenho de acordo com os especialistas.	138
Tabela 20: Nível de concordância dos especialistas em relação aos objetivos representados na ontologia.	139
Tabela 21: Nível de concordância dos especialistas em relação aos entregáveis representados na ontologia.	139
Tabela 22: Nível de concordância dos especialistas em relação às atividades da ontologia (parte 1).	140
Tabela 23: Nível de concordância dos especialistas em relação às atividades da ontologia (parte 2).	141
Tabela 24: Nível de concordância dos especialistas em relação ao agrupamento das atividades na ontologia.	142
Tabela 25: Comentários adicionais dos participantes sobre as perguntas abertas.	142
Tabela 26: Semelhanças e diferenças entre os objetivos dos especialistas e os objetivos da ontologia proposta.	143
Tabela 27: Semelhanças e diferenças entre os entregáveis dos especialistas e os existentes na ontologia.	145
Tabela 28: Semelhanças e diferenças entre as ferramentas dos especialistas e as ferramentas da ontologia.	146
Tabela 29: Semelhanças e diferenças entre as atividades dos especialistas e as atividades da ontologia.	148
Tabela 30: Semelhanças e diferenças entre as métricas dos especialistas e as métricas da ontologia proposta.	150

LISTA DE SIGLAS

API	Application Programming Interface
ARP	Address Resolution Protocol
DL	Description Logic
DNS	Domain Name System
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IMAP	Internet Message Access Protocol
IP	Internet Protocol
IRI	Internationalized Resource Identifier
JMS	Java Message Service
JUNG	Java Universal Network/Graph
NFS	Network File System
OWL	Web Ontology Language
PDF	Portable Document Format
PT	Performance Test
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SPE	Software Performance Engineering
SQWRL	Semantic Query-enhanced Web Rule Language
SSH	Secure Shell
SWEBOK	Software Engineering Body of Knowledge
SWRL	Semantic Web Rule Language
TCP	Transmission Control Protocol
TPS	Transactions Per Second
UDP	User Datagram Protocol
UML	Unified Modeling Language
XML	Extensible Markup Language

SUMÁRIO

1.	INTRODUÇÃO	21
1.1.	Objetivos	22
1.2.	Organização	23
2.	FUNDAMENTAÇÃO TEÓRICA	25
2.1.	Teste de Software	25
2.2.	Teste de Desempenho de Software	28
2.3.	Ontologia	30
2.4.	Protégé.....	33
3.	TRABALHOS RELACIONADOS	35
3.1.	Ontologias e Desempenho de Software	37
3.2.	Ontologias e Teste de Software	47
4.	ONTOLOGIA PROPOSTA	57
4.1.	Escopo da Ontologia	57
4.2.	Conceitos, Propriedades e Instâncias da Ontologia	60
5.	AVALIAÇÃO DA ONTOLOGIA	79
5.1.	Comparação com Ontologias Relacionadas	80
5.2.	Avaliação da Ontologia por Especialistas	85
6.	APLICAÇÕES DA ONTOLOGIA PROPOSTA.....	93
6.1.	Aplicação Desenvolvida como Plugin do Protégé.....	93
6.2.	Execução de Consultas SQWRL sobre a Ontologia	98
6.3.	Utilização de Reasoners sobre a Ontologia	100
6.4.	Utilização da Ontologia no WebProtégé	102
6.5.	Aplicação Desenvolvida Baseada na Ontologia	104
6.5.1.	Utilização da OWL API.....	105
6.5.2.	Exemplo de Uso da iText API	107
7.	CONSIDERAÇÕES FINAIS	109
	REFERÊNCIAS	117
	APÊNDICE A – QUESTIONÁRIO SOBRE TESTE DE DESEMPENHO	123
	APÊNDICE B – RESPOSTAS DOS ESPECIALISTAS.....	131
	APÊNDICE C – COMPARAÇÃO DO QUESTIONÁRIO COM A ONTOLOGIA	143

1. INTRODUÇÃO

“We live in a society exquisitely dependent on science and technology, in which hardly anyone knows anything about science and technology.” (Carl Sagan)

Os profissionais da computação perceberam a necessidade de desenvolver técnicas de teste de software em meados da década de 1980 [24]. Contudo, foi apenas na década de 1990 que começaram a aparecer *whitepapers*, seminários e artigos nesta área [24]. Conseqüentemente, o conhecimento em teste de software ainda está em fase de expansão e consolidação. O teste é uma tarefa não trivial por causa das características próprias dos softwares, como intangibilidade e complexidade [4]. A dificuldade em testar software é caracterizada, sobretudo, devido à falta de profissionais especializados; à dificuldade em implantar um processo de teste; ao desconhecimento de técnicas e procedimentos de teste e ao desconhecimento sobre como planejar os testes [4]. Alinhado a isto, as atividades de planejamento e elaboração dos testes de software demandam testadores com conhecimento sobre as técnicas de teste, critérios, artefatos e ferramentas [18]. Esta diversidade de conceitos levanta uma questão: como estabelecer um entendimento compartilhado sobre teste de software? Ontologias são ferramentas capazes de capturar o conhecimento de domínios [54] e, por este motivo, mostram potencial para preencher as lacunas citadas no processo de teste [61]. O tema ontologia é um tópico de pesquisa popular em diferentes comunidades como engenharia do conhecimento, processamento da linguagem natural e gestão do conhecimento [15]. Em Inteligência Artificial, ontologias são desenvolvidas para facilitar a aquisição, organização, representação, compartilhamento e reuso do conhecimento de um domínio [15] [19]. De acordo com as motivações apresentadas, este trabalho tem como objetivo propor uma ontologia para modelar os conhecimentos sobre o domínio de teste de desempenho.

Visando explorar os temas essenciais deste estudo, os tópicos iniciais apresentam uma fundamentação teórica sobre ontologias e sobre teste de software. Em seguida, são apresentadas sínteses dos trabalhos relacionados de modo a identificar e comparar suas principais contribuições na área. A revisão da literatura também possui como objetivo tornar claro de que forma ontologias foram ou podem ser aplicadas nos domínios de teste de software e teste de desempenho. Uma das aplicações se refere ao uso de ontologias no planejamento dos testes de desempenho, sendo um dos incentivos para construir uma ontologia neste domínio. Devido à complexidade inerente aos testes de desempenho, o

indicado é que eles sejam planejados por um time de testadores experientes. Em outras palavras, elaborar um teste de desempenho exige conhecimentos de teste encontrados em testadores especializados, e não apenas em um único indivíduo [24]. Portanto, este trabalho propõe uma ontologia para modelar os conhecimentos sobre o domínio de teste de desempenho com o objetivo de apoiar a decisão de testadores, em especial aqueles que ainda não possuem a experiência necessária para planejar os testes de desempenho.

A ontologia foi representada na linguagem OWL (Web Ontology Language) [59], desenvolvida no editor Protégé [62] e seguindo a metodologia de Noy e McGuinness [49]. As aplicações da ontologia exploradas nesta dissertação foram direcionadas a auxiliar os testadores quanto ao planejamento de seus testes. Uma das aplicações foi desenvolvida como plugin para o Protégé [62], usando a API oferecida pelo Protégé [62] para manipular a ontologia. Foi desenvolvida também uma aplicação em Java [55] que usa a OWL API [45] para manipular a ontologia. Além disso, foi investigada a utilização do *reasoner* Pellet [21] para inferência de novos axiomas com base na ontologia e foi explorada a utilização da linguagem SQWRL (Semantic Query-Enhanced Web Rule Language) [47] para executar consultas na ontologia. Por fim, a ontologia proposta é comparada com duas ontologias relacionadas e também passa por um processo de avaliação por especialistas em teste de desempenho.

1.1. Objetivos

Esta pesquisa tem como objetivo geral a construção de uma ontologia sobre teste de desempenho de software. Um dos objetivos da ontologia será fornecer uma base de conhecimento sobre os conceitos mais relevantes do domínio como, por exemplo, métricas de desempenho, ferramentas, atividades, objetivos de teste, etc. Embora a utilização desta base de conhecimento possa ser variada, a pesquisa se concentrou na aplicação da ontologia nas atividades de planejamento e documentação dos testes de desempenho. Embora existam ontologias sobre teste de software, como a *OntoTest (Ontology of Software Testing)* [19] e a *SwTO (Software Test Ontology)* [13], não foram encontrados trabalhos propondo ontologias sobre o domínio de teste de desempenho. Portanto, este trabalho teve como base os seguintes objetivos:

- Investigar aplicações de ontologias no teste de software e no teste de desempenho.
- Pesquisar tecnologias utilizadas na construção e utilização de ontologias, em especial o Protégé, a linguagem OWL e recursos relacionados.

- Desenvolver e explorar aplicações baseadas na ontologia proposta.
- Investigar e aplicar técnicas para avaliar a ontologia proposta.
- Comparar a ontologia proposta com ontologias relacionadas.
- Comparar as principais contribuições entre este trabalho e os trabalhos relacionados.

1.2. Organização

Este trabalho está organizado em sete capítulos. O capítulo 1 introduziu uma visão geral sobre a contextualização, a motivação e os objetivos desta pesquisa. O capítulo 2 é responsável pela fundamentação teórica sobre teste de software, teste de desempenho e ontologias. Em seguida, o capítulo 3 apresenta um estudo sobre trabalhos relacionados que definem ou aplicam ontologias sobre o domínio de teste de software. Os conceitos, propriedades, axiomas e instâncias da ontologia proposta são explicados na seção 4. A comparação da ontologia proposta com duas relacionadas e a opinião dos especialistas em teste de desempenho sobre a ontologia é detalhada na seção 5. Os softwares desenvolvidos que utilizam esta ontologia são apresentados na seção 6, juntamente com exemplos de uso. As considerações finais da pesquisa são apresentadas na seção 7, onde as principais contribuições desta dissertação são destacadas e comparadas com as contribuições dos trabalhos relacionados.

2. FUNDAMENTAÇÃO TEÓRICA

Para facilitar o entendimento e unificar as definições adotadas no desenvolvimento deste trabalho, as subseções seguintes apresentam uma fundamentação teórica sobre teste de software, teste de desempenho e ontologias.

2.1. Teste de Software

“Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence.” (Edsger Dijkstra)

O teste é utilizado para avaliar a qualidade de um produto e possibilitar sua melhoria por meio da identificação de problemas e defeitos [1]. O teste de software pode ser definido como o processo de executar um software de uma maneira controlada com o objetivo de avaliar se o mesmo se comporta conforme o especificado [4]. Segundo [27], o teste é o processo de analisar um item de software para avaliar suas características e detectar diferenças entre as condições existentes e as condições especificadas. O teste de software engloba questões relacionadas ao processo, às técnicas, às ferramentas, ao ambiente e aos artefatos [2]. Além disso, o alvo do teste de software pode variar desde um módulo em singular, um grupo de módulos ou o sistema inteiro. Deste modo, alguns autores [1] [25] dividem as etapas de teste em três: unitário, de integração e de sistema. O desempenho é avaliado pelo teste de sistema, que é responsável pela avaliação também de outros requisitos não funcionais (como segurança e confiabilidade). Segundo o Corpo de Conhecimento em Engenharia de Software (SWEBOK) [1], o teste de software consiste na verificação **dinâmica** do comportamento de um programa por meio de um conjunto **finito** de casos de teste. Estes casos de teste são **selecionados** de um domínio geralmente infinito de possíveis execuções. Finalmente, os resultados obtidos nos testes devem ser comparados com um dos seguintes resultados **esperados**:

- a especificação do software (referido como testes para verificação);
- as expectativas dos usuários (referido como testes para validação);
- o comportamento previsto de requisitos implícitos ou expectativas razoáveis.

Por este motivo, o teste está estreitamente relacionado com os requisitos, uma vez que os requisitos expressam necessidades ou restrições em um software. Além desta

definição, o *SWEBOK* [1] define requisito de software como uma propriedade que deve ser exibida pelo software a fim de resolver algum problema do mundo real. Os requisitos podem ser classificados em **funcionais** e **não funcionais** [1]. Os requisitos funcionais descrevem funções que o software deve executar, ou seja, “o que” deve ser feito, por exemplo: cadastrar usuário, efetuar transferência bancária e enviar mensagem. Os requisitos não funcionais tratam de questões como desempenho, segurança, confiabilidade, usabilidade, acessibilidade e manutenibilidade. Em outras palavras, os requisitos não funcionais expressam restrições sobre qualidades de comportamento do sistema. É importante ressaltar que tanto os requisitos funcionais quanto os requisitos não funcionais podem causar a inutilização do sistema se não forem atendidos.

Segundo [4], os softwares apresentam características como intangibilidade e complexidade, o que torna o teste uma tarefa não trivial. A dificuldade em testar software é caracterizada principalmente devido aos seguintes pontos [4]:

- o teste de software é um processo caro;
- existe uma falta de conhecimento sobre a relação custo/benefício do teste;
- há falta de profissionais especializados na área de teste;
- existem dificuldades em implantar um processo de teste;
- há o desconhecimento de um procedimento de teste adequado;
- há o desconhecimento de técnicas de teste adequadas;
- há o desconhecimento sobre como planejar a atividade de teste; e
- a preocupação com a atividade de teste somente na fase final do projeto.

Os casos de teste podem ser projetados visando diferentes objetivos, como revelar divergências entre os requisitos dos usuários; avaliar a conformidade do software com uma especificação padrão; analisar sua robustez em condições estressantes de carga, e assim por diante [3]. Segundo o *SWEBOK* [1], o planejamento do teste de software deve começar nos estágios iniciais do processo de requisitos e as atividades de teste devem estar presentes em todo o processo de desenvolvimento e manutenção. O *SWEBOK* conclui que a prevenção é a atitude mais correta em direção à qualidade, ou seja, é melhor evitar problemas do que corrigi-los. Neste contexto, o teste, por meio da descoberta de falhas, deve verificar até que ponto a prevenção está sendo efetiva.

Os termos mais importantes relacionados ao teste de software, de acordo com o glossário da Engenharia de Software definido pela IEEE [28], estão listados a seguir:

- **test bed**: ambiente contendo os hardwares, instrumentação, simuladores, ferramentas de software e demais elementos necessários à condução do teste.

- **test case:** um conjunto de entradas do teste, condições de execução e resultados esperados. Um caso de teste é desenvolvido para um objetivo particular como, por exemplo, exercitar um caminho de um programa ou verificar um requisito específico.
- **test case generator:** uma ferramenta de software que recebe como entrada código fonte, critérios de teste, especificações ou definições de estruturas de dados e gera como saída dados de entradas do teste e resultados esperados.
- **test case specification:** um documento que especifica as entradas do teste, condições de execução e resultados previstos para um item a ser testado.
- **test criteria:** os critérios que um sistema ou componente deve satisfazer para passar em um dado teste.
- **test design:** documentação que especifica os detalhes da abordagem de teste para um recurso de software e identifica os testes associados.
- **test documentation:** documentação descrevendo planos para, ou resultados de, o teste de um sistema ou componente. A documentação de teste inclui especificação dos casos de teste, relatório de incidente de teste, registro de teste, plano de teste, procedimento de teste e relatório de teste.
- **test incident report:** documento que descreve um evento ocorrido durante o teste e que requer uma investigação mais aprofundada.
- **test item:** um item de software que é um objeto do teste.
- **test item transmittal report:** documento que identifica um ou mais itens submetidos ao teste, contendo o status atual e informações de localização.
- **test log:** registro cronológico dos detalhes relevantes sobre a execução de um teste.
- **test objective:** conjunto de características do software a serem medidas em condições especificadas, para que o comportamento real seja comparado com o comportamento requerido descrito na documentação do software.
- **test phase:** período de tempo no ciclo de vida de software em que os componentes de um produto de software são integrados e avaliados para determinar se os requisitos estão sendo satisfeitos.
- **test plan:** documento que descreve a abordagem técnica e administrativa a ser seguida para testar um sistema ou componente. O plano de teste identifica os itens a serem testados, tarefas e responsáveis pela execução de cada tarefa, cronogramas e recursos necessários para a execução do teste.
- **test procedure:** descreve instruções detalhadas para a configuração, execução e avaliação dos resultados de um dado caso de teste. O procedimento de teste é uma

documentação usada para especificar uma sequência de ações essenciais para a execução de um teste (sinônimo de script de teste).

- **test report:** documento que descreve a execução e os resultados de um teste realizado em um sistema ou componente.
- **test summary report:** documento que resume as atividades de teste e os resultados, contendo também uma avaliação dos itens de teste relacionados.
- **testing:** processo de operar um sistema ou componente dentro de condições específicas, observando ou gravando resultados e fazendo uma avaliação de algum aspecto do sistema ou componente. Também pode ser definido como o processo responsável por analisar características dos itens de software com o objetivo de detectar diferenças entre as condições existentes e as requeridas.

2.2. Teste de Desempenho de Software

“The word performance in computer performance means the same thing that performance means in other contexts, that is, it means *How well is the computer doing the work it is supposed to do?*” (Arnold Allen)

O desempenho é uma qualidade dos softwares que é afetado desde suas camadas inferiores, como sistema operacional, middleware, hardware e redes de comunicação [48]. Um sistema possui bom desempenho quando apresenta um tempo de resposta adequado e aceitável, mesmo se submetido a um volume de processamento próximo de situações reais ou de pico [2]. A área responsável pelo estudo do desempenho é conhecida Engenharia de Desempenho de Software. De acordo com Woodside, Franks e Petriu [48], a Engenharia de Desempenho de Software é o campo responsável pelo estudo de atividades de Engenharia de Software e análises relacionadas usadas ao longo do ciclo de desenvolvimento de software e direcionadas aos requisitos de desempenho. Segundo estes autores, existem duas abordagens possíveis dentro da Engenharia de Desempenho de Software: baseada em **medição** ou baseada em **modelos**. Esta dissertação possui como foco a abordagem de medição, incluindo, portanto, atividades de teste, diagnóstico e *tuning*. Estas atividades são realizadas apenas no final do ciclo de desenvolvimento, dado que o sistema real precisa ser executado e medido. A outra abordagem de avaliação de desempenho é baseada em modelos, onde os resultados quantitativos dos modelos são usados para ajustar a arquitetura e o projeto com o objetivo de atingir os requisitos de

performance. Tendo conceituado essa área de estudo, podemos entender melhor a definição de teste de desempenho dada a seguir.

Segundo [36], o teste de desempenho determina ou valida a capacidade de resposta (*responsiveness*), vazão (*throughput*), confiabilidade (*reliability*) e/ou escalabilidade (*scalability*) de um sistema sob uma dada carga (*workload*). O teste de desempenho geralmente tem o objetivo de [36]: definir as características de desempenho do sistema; encontrar problemas de desempenho, *i.e.*, gargalos (*bottlenecks*); permitir a otimização do sistema; estimar uma configuração de hardware adequada para a aplicação; entre outros. Além disso, o teste de desempenho abrange as seguintes atividades [36]:

- 1) Identificação do ambiente de teste.** Além do ambiente físico de teste e do ambiente de produção, devem ser identificados os recursos e as ferramentas disponíveis à equipe de testes. O ambiente físico inclui configurações de hardware, software e rede.
- 2) Identificação dos critérios de aceitação de desempenho.** Quais os objetivos e as restrições de tempo de resposta, vazão e utilização de recursos. Em geral, tempo de resposta é uma preocupação (*concern*) do usuário, vazão é uma preocupação do negócio e utilização de recursos é uma preocupação no nível do sistema.
- 3) Planejamento e projeto dos testes.** Identificar cenários principais, determinar a variabilidade entre os usuários representativos e como simular esta variabilidade, definir os dados do teste e estabelecer métricas para coletar.
- 4) Configuração do ambiente de teste.** Preparação do ambiente de testes, ferramentas e recursos necessários para executar o teste.
- 5) Implementação dos testes.** Desenvolvimento dos testes de desempenho de acordo com o planejamento.
- 6) Execução do teste.** Execução do teste e monitoramento.
- 7) Análise dos resultados, relatórios e retestagem.** Quando todos os valores monitorados estiverem dentro dos limites aceitáveis, o teste finalizou para aquele cenário em particular e para aquela configuração específica.

O teste de desempenho pode acabar recebendo diferentes classificações de acordo com seus objetivos, *e.g.*, teste de resistência, teste de pico ou outros. Por exemplo, segundo [36], **teste de carga** é uma subcategoria do teste de desempenho focada em avaliar o desempenho do sistema quando submetido a uma carga equivalente a de produção. Por outro lado, um **teste de estresse** representa outra subcategoria de teste de desempenho focada em avaliar o desempenho do sistema quando submetido a uma carga superior a de produção. Este último tipo de teste também pode avaliar o sistema

sob condições estressantes como insuficiência de espaço em disco, limitação de memória, etc. O teste de estresse deve ser projetado para avaliar sob quais condições uma aplicação vai falhar, como será essa falha e quais indicadores podem ser monitorados para avisar que uma falha está prestes a ocorrer.

Esta seção introduziu os principais conceitos relacionados ao teste de software e ao teste de desempenho. Dada a riqueza de conhecimentos da área de testes que os testadores devem possuir, este trabalho se propõe a investigar como ontologias podem auxiliá-los a representar estes conhecimentos. Ontologia é uma técnica de representação de conhecimento atualmente considerada o estado da arte em Inteligência Artificial. Assim, a próxima seção apresenta uma fundamentação teórica sobre ontologias, com o objetivo de fornecer uma visão geral sobre definições, aplicações, metodologias e ferramentas envolvidas no desenvolvimento e utilização de ontologias.

2.3. Ontologia

“Our everyday reasoning is not precise, but it is nevertheless efficient. Nature, itself, from galaxies to genes, is approximate and inexact. Philosophical concepts are among the least precise. Terms such as ‘mind’, ‘perception’, ‘memory’, and ‘knowledge’ do not have either a fixed nor a clear meaning but they make sense just the same.” (Gian-Carlo Rota)

Uma ontologia é geralmente utilizada como uma estrutura que captura o conhecimento sobre uma determinada área, fornecendo conceitos e relacionamentos relevantes [32]. A ontologia de um domínio abrange sua terminologia (ou vocabulário), os conceitos fundamentais, sua classificação, taxonomia, relações e axiomas [16]. O autor Thomas Gruber [65] define uma ontologia como uma “*especificação explícita de uma conceitualização*”. Porém, o autor Willem Borst, em sua tese de doutorado [73], modificou a definição anterior ao dizer que ontologia é uma “*especificação formal de uma conceitualização compartilhada*”. O termo *conceitualização* faz referência ao modelo abstrato de algum fenômeno identificado no mundo por ter conceitos relevantes. *Formal* refere-se ao fato de que a ontologia deve ser legível por máquina, enquanto que *compartilhada* significa que o conhecimento representado na ontologia é consensual (aceito por um grupo). Segundo Noy e McGuinness [49], ontologias descrevem um domínio de discurso utilizando classes, propriedades, relacionamentos, atributos e restrições de conceitos. Ainda, uma ontologia e suas instâncias individuais de classes

constituem uma *base de conhecimento* [49]. Seguindo essa mesma definição, de acordo com [54], uma ontologia deve possuir, no mínimo, os seguintes componentes:

- **Classes**, que representam conceitos. As classes podem ser organizadas em taxonomias com o uso de herança. A linguagem ontológica usada pode permitir a definição de atributos ou propriedades em classes.
- **Relacionamentos**, que representam um tipo de interação ou associação entre conceitos do domínio. Um relacionamento é como uma função matemática, onde o domínio e a imagem são conceitos. Os atributos das classes são relacionamentos unários, pois possuem como domínio um conceito e a imagem é formada por um tipo de dado (número, data, string).
- **Axiomas**, que modelam sentenças que são sempre verdade. Os axiomas são úteis na inferência de novos conhecimentos e podem ser construídos em lógica de primeira ordem.
- **Instâncias**, que representam elementos de um dado conceito do domínio. Segundo [52], podem existir **fatos** representando relações entre instâncias.

Além de capturar o conhecimento sobre um domínio de interesse, segundo [49], ontologias podem ser aplicadas para: compartilhar um entendimento comum sobre a estrutura da informação entre pessoas e softwares; possibilitar análise e reuso do conhecimento do domínio; tornar explícitas as suposições sobre um domínio; e criar uma separação entre o conhecimento operacional e o conhecimento do domínio.

Para se trabalhar com ontologias é necessário antes representá-las em uma linguagem ou formalismo. Diferentes linguagens ontológicas são comparadas em [52] quanto às suas capacidades de representação do conhecimento e inferência. O domínio de conhecimento é formado pelos componentes ontológicos citados anteriormente. Por fim, o processo de raciocínio proporcionado por uma ontologia é executado sobre o seu domínio de conhecimento [52]. São exemplos de linguagens ontológicas: Ontolingua [6], OCML [20], OWL [17], LOOM [58], F-Logic [46] e OIL [14].

Sobre a construção de ontologias, segundo [49], não existe uma forma única e correta de modelar um domínio, ou seja, sempre existem alternativas viáveis e a melhor solução quase sempre depende da aplicação em mente. Além disso, o desenvolvimento de ontologias é inevitavelmente um processo iterativo. Segundo essas premissas, em [49] foi elaborada uma metodologia para desenvolvimento de uma ontologia:

- 1) Determinar o domínio e o escopo da ontologia;
- 2) Considerar o reuso de ontologias existentes;

- 3) Enumerar termos importantes na ontologia;
- 4) Definir classes e hierarquia de classes;
- 5) Definir propriedades das classes;
- 6) Definir as características (*facets*) das propriedades;
- 7) Criar instâncias.

Outra metodologia para desenvolvimento de ontologias foi proposta em [29] e está ilustrada na Figura 1. As atividades descritas em ambas as metodologias possuem semelhanças, contudo a metodologia apresentada em [49] utiliza uma linguagem de menor abstração, pois usa termos como classes, propriedades e instâncias. Outra diferença identificada foi a presença, na metodologia definida em [29], de atividades específicas para a escolha da linguagem e das ferramentas de desenvolvimento.

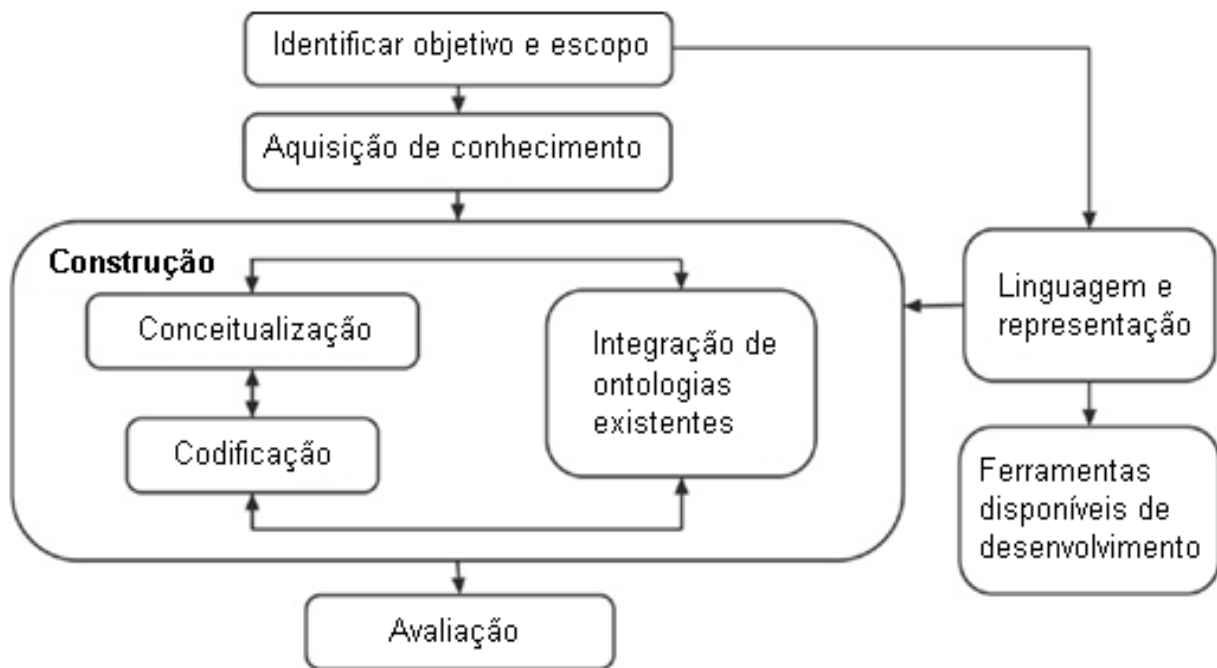


Figura 1: Ciclo de vida de construção de uma ontologia (Fonte [29]).

Com relação às ferramentas para desenvolvimento de ontologias, um estudo que compara suas características gerais e capacidades de representação do conhecimento pode ser encontrado em [5]. As ferramentas analisadas neste estudo incluem Ontolingua [6], *WebOnto* [33], *Protégé* [35] [50] e *ODE* [44], sendo o *Protégé* uma das ferramentas mais difundidas e robustas. Assim sendo, o foco da subseção seguinte é apresentar a ferramenta *Protégé*. Mais informações sobre o estado da arte em relação às ferramentas, metodologias e linguagens para desenvolvimento de ontologias estão disponíveis nas referências [8], [53], [54], e [72].

2.4. Protégé

“Artificial intelligence is the study of how to make real computers act like the ones in the movies.” (Anonymous)

O Protégé é um ambiente para apoio ao desenvolvimento de sistemas baseados em conhecimento [35]. Este software *open source*, desenvolvido na Universidade de Stanford e aplicado inicialmente no domínio da medicina, pode ser utilizado para criar e editar ontologias e bases de conhecimento [62]. O Protégé implementa um conjunto de estruturas de modelagem de conhecimento e ações que suportam a criação, a visualização e a manipulação de ontologias representadas em diversos formatos. Assim, esta ferramenta suporta dois diferentes paradigmas referentes a modelagem de ontologias: baseado em frames ou em OWL [62].

- O editor **Protégé-Frames** possibilita aos usuários a criação e o povoamento de ontologias no formato *frame-based*. Neste modelo, uma ontologia consiste em um conjunto de classes organizadas em uma hierarquia para representar os conceitos do domínio; um conjunto de *slots* associados às classes que representam suas propriedades e relacionamentos; e as instâncias das classes.
- O editor **Protégé-OWL** permite aos usuários a construção de ontologias para a web semântica, em particular usando OWL (Web Ontology Language). Uma ontologia em OWL pode incluir descrições de classes, propriedades e suas instâncias.

Para o desenvolvimento da ontologia proposta neste trabalho, o editor Protégé-OWL foi adotado por ser a versão mais atual da ferramenta e com suporte a um número maior de recursos quando comparado ao Protégé-Frames. A versão 4.2 foi utilizada por ser a versão mais atual até o momento (foi disponibilizada em outubro de 2011 [62]). Como ilustrado na Figura 2, a interface gráfica da versão 4.2 é dividida em abas, sendo cada aba responsável por determinados aspectos do desenvolvimento da ontologia. A Figura 2 exibe as informações da aba “Classes” (é importante observar que classes e conceitos são sinônimos no contexto do Protégé).

Tendo apresentado uma visão geral sobre ontologias e o editor Protégé, a seção seguinte apresentará aplicações propostas na literatura referentes ao uso de ontologias no teste ou no desempenho de software.

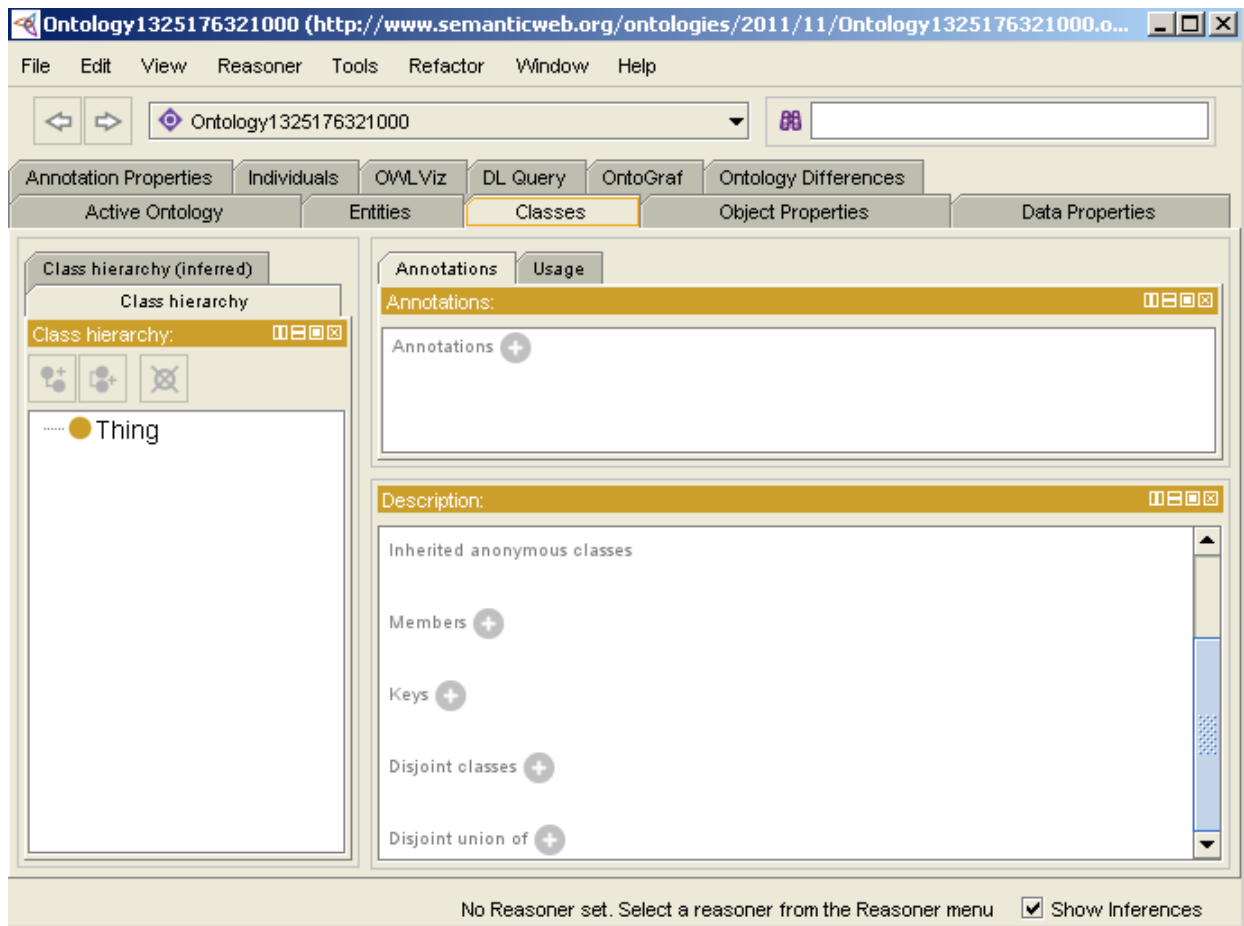


Figura 2: Interface gráfica do Protégé versão 4.2.

3. TRABALHOS RELACIONADOS

“If I have seen further it is by standing on the shoulders of giants.” (Isaac Newton)

O objetivo desta seção é fornecer uma visão geral do estado da arte em relação ao uso de ontologias no teste de software e no desempenho de software. Assim, serão apresentadas comparações e sínteses dos trabalhos relacionados considerados os mais relevantes na literatura analisada. O critério de inclusão adotado foi que o trabalho deve definir ou utilizar ontologias no domínio de teste de software ou desempenho de software. A seguir são destacados os principais argumentos dos autores quanto à aplicação de ontologias no domínio de teste. Uma vantagem encontrada em [29] para justificar o uso de ontologias sobre o desempenho de software se refere à habilidade de raciocínio, que possibilita a tomada de decisões sobre o desempenho e outras características não funcionais do sistema. Em outras palavras, ontologias permitem uma interpretação lógica e a aplicação de regras de inferência sobre a representação semântica de informações do domínio de desempenho de software. Na visão de [54] e [61], ontologias são ferramentas capazes de capturar o conhecimento de domínios em um formato compreensível por máquinas e, por este motivo, mostram potencial de serem usadas na automação do processo de teste. Além de fornecer um vocabulário para representar e comunicar o conhecimento sobre o teste de software, tal ontologia deve apresentar um conjunto de relacionamentos existentes entre os conceitos deste domínio [39]. Ontologias podem contribuir na área de teste de software, pois [39]:

- Fornecem uma fonte de definição precisa de termos que podem ser usados na comunicação entre testadores, desenvolvedores, gerentes e usuários.
- Oferecem um entendimento consensual compartilhado do teste de software.
- Tornam explícitas as suposições ocultas relacionadas aos objetos pertencentes ao conhecimento de teste de software.

Segundo a literatura analisada, a seguir são identificadas as aplicações baseadas em ontologias. Tais aplicações são bastante diversificadas quanto ao uso de ontologias nos domínios de teste de software ou desempenho de sistemas computacionais. Algumas destas aplicações já chegaram a ser desenvolvidas e validadas, contudo outras ainda estão na fase conceitual. As principais áreas de estudo identificadas são as seguintes:

Interoperabilidade entre os softwares para teste de software. Diferentes softwares podem ser usados para criar, executar e analisar testes de desempenho, porém

cada uma dessas ferramentas estrutura as informações de maneiras distintas. Um nível maior de interoperabilidade entre as ferramentas para teste de desempenho poderia ser alcançado se as ferramentas compartilhassem a mesma ontologia. Neste caso, seria possível especificar o teste em uma ferramenta qualquer e executá-lo usando quaisquer outras ferramentas sem necessidade de alterar o teste projetado. Além disso, os resultados monitorados durante a execução do teste poderiam ser carregados e analisados usando quaisquer ferramentas que compartilhassem a mesma ontologia. Esta vantagem de interoperabilidade foi comentada em [19] e [69]. Além disso, uma ontologia de teste de software pode auxiliar no estabelecimento de uma arquitetura de referência, o que facilitaria o uso e a integração de ferramentas, processos e artefatos [19].

Sugestão e validação do planejamento dos testes. De acordo com a configuração de um sistema que se deseja testar o desempenho, como são escolhidas as tecnologias que podem ser usadas? Por exemplo, se desejamos monitorar o sistema operacional Linux, quais ferramentas podem ser usadas? Estas perguntas poderiam ser respondidas por uma ontologia que represente o conhecimento sobre tecnologias (técnicas, processos ou ferramentas) utilizadas no teste de desempenho. Para isso, é necessário representar informações relacionadas as restrições e as capacidades envolvidas na aplicação de cada tecnologia, e.g., uma capacidade fornecida pela ferramenta LoadRunner é conseguir testar aplicações sobre o protocolo HTTP. Dessa forma, uma vez que o ambiente de teste fosse definido, seria possível escolher quais tecnologias podem ser usadas no teste de desempenho. Além disso, de acordo com as escolhas selecionadas, pode ser gerado um plano de teste correspondente. Uma referência relevante para definir a estrutura do plano de teste é, por exemplo, o Standard 829 da IEEE [27]. O plano de teste é um documento que fornece a base para a execução dos testes de software de modo organizado, sistemático e bem documentado [27]. O uso de ontologias para o planejamento e a especificação de testes foi abordado como aplicação teoricamente viável em [61].

Consulta semântica na documentação de testes. O uso de ontologias sobre documentos de testes permite que sejam executadas consultas semânticas sobre as informações dos testes. O exemplo dado por [61] é que seria possível perguntar quais classes e métodos não foram suficientemente testadas após a execução de um teste de cobertura de código. A referência [41] propõem uma técnica para recuperação de documentos sobre teste de software com base em ontologias. A documentação de teste pode ser construída com base no padrão *IEEE Standard for Software and System Test Documentation* [27], que descreve o conteúdo de cada documento de teste.

Geração de casos de teste. Alguns dos trabalhos analisados (e.g. [61], [71], [70] e [67]) comentam sobre a possibilidade de geração de casos de teste para uma dada aplicação a partir da combinação de duas ontologias: uma sobre teste de software e uma sobre o domínio da aplicação a ser testada. Uma abordagem para geração de testes unitários baseada em ontologias é apresentada em [71], onde é argumentado como próximos passos a expansão da técnica nas fases de teste de integração e de sistema. Por outro lado, em [42] foi explicado como gerar dados para testes de robustez aplicando uma ontologia que abrange o domínio dos serviços *web* a serem testados.

Melhoria do desempenho durante a execução do sistema. A melhoria do desempenho durante a execução do sistema foi citada, por exemplo, em [29] e [10]. Estes trabalhos utilizam ontologias para representar informações do contexto atual do sistema e usam medições de desempenho ao longo do tempo para alterar a configuração do sistema para tentar melhorar o desempenho. Por exemplo, em [29], o raciocínio sobre o desempenho é baseado na ontologia e em regras definidas em lógica de primeira ordem.

Aprendizagem e padronização do vocabulário sobre teste de software. Alguns autores, e.g. [56] e [19], justificam que uma ontologia sobre teste de software pode ser utilizada para padronizar o vocabulário da área e facilitar a troca de conhecimentos. Segundo [19], também é possível usar tal ontologia para aprendizagem sobre o domínio. Nesta direção, as ontologias também oferecem um entendimento compartilhado sobre os conceitos e relacionamentos existentes no teste de desempenho.

Após apresentar resumidamente quais aplicações de ontologias foram propostas na área de teste, as duas subseções seguintes apresentam uma síntese de cada trabalho. Estas sínteses possuem como foco executar uma análise individual da abordagem e dos resultados de cada um dos trabalhos relacionados.

3.1. Ontologias e Desempenho de Software

“Artificial intelligence is that field of computer usage which attempts to construct computational mechanisms for activities that are considered to require intelligence when performed by humans.” (Derek Partridge)

Um dos objetivos desta pesquisa é avaliar como técnicas de representação do conhecimento (*i.e.*, ontologias) podem auxiliar nas atividades de teste. Para identificar as aplicações comentadas na literatura, o tópico a seguir resume trabalhos que abordam

simultaneamente os assuntos ontologias e desempenho de software, destacando os pontos principais de cada trabalho analisado e comparando as aplicações identificadas.

How far are we from the definition of a common software performance ontology [69]. Segundo Vittorio Cortellessa [69], a definição de um padrão para representar as informações relacionadas ao desempenho dos artefatos de software resultaria em vantagens de interoperabilidade de ferramentas e transformação de modelos. Porém, tal padrão ainda não foi definido. Em [69] são analisados os três metamodelos de desempenho de software listados a seguir:

- UML Profile for Schedulability, Performance and Time (SPT): abordagem proposta pelo Object Management Group (OMG) para representar dados de desempenho em UML.
- Core Scenario Model (CSM): abordagem desenvolvida pelo grupo de Sistemas Distribuídos e de Tempo Real da Universidade de Carleton para integrar anotações de desempenho em modelos de software.
- Software Performance Engineering Meta-Model (SPE-MM): abordagem criada por Connie Smith, Lloyd Williams e Catalina Lladó para definir entidades e relacionamentos úteis na construção de Software Execution Models e System Executions Models.

Além disso, na referência [69] são idealizadas duas abordagens para a criação de uma ontologia neste domínio: (i) *bottom-up*, que extrai conhecimento dos metamodelos; e (ii) *top-down*, que é dirigida por um conjunto de requisitos. O desempenho de software inclui conceitos como medição de desempenho, monitoramento, gerenciamento e geração de carga. Porém, o trabalho de Vittorio [69] foca em aspectos de modelagem do desempenho de software e por este motivo não considera estes conceitos citados anteriormente. As entidades de cada um dos três metamodelos analisados são agrupadas em uma das seguintes categorias:

- Software Behavior: entidades que representam as dinâmicas do software.
- Resources: entidades que representam os recursos computacionais (exemplo: espaço de armazenamento e poder de processamento).
- Workload: entidades que representam a carga.

Além de comparar os conceitos comuns nos modelos SPT, CSM e SPE-MM, este trabalho definiu características que deveriam existir em uma ontologia de desempenho de software no caso de construí-la usando a abordagem *top-down*. Segundo [69], uma ontologia de desempenho de software deveria:

- **Ser capaz de representar os resultados da análise de desempenho.** Os resultados obtidos na análise devem ser representados de uma forma única, independente do tipo

de modelo e solução adotada. Este requisito foi abordado parcialmente na UML SPT, onde entidades foram introduzidas para representar explicitamente índices de interesse (tempo de resposta, utilização) e suas representações matemáticas (média, distribuição cumulativa). A análise deve ser capaz de relacionar resultados de desempenho (tempo de resposta, utilização) para fornecer recomendações aos engenheiros de software.

- **Lidar com o maior número de formalismos de desempenho quanto possível.** O conjunto de notações de desempenho geralmente adotados (filas de rede de espera, redes de Petri estocásticas) suportam a viabilidade deste requisito. Ou seja, a conversão de modelos UML anotados em outros formalismos (e. g., cadeia de Markov).
- **Prover formas de integrar facilmente um modelo de software com anotações de desempenho.** Hoje, raciocinamos sobre modelos UML, mas tal ontologia não pode ser adaptada em torno de uma linguagem de modelagem de software, caso contrário ela não seria capaz de representar “a natureza e as relações do ser”.
- **Ser compatível com representações internas das ferramentas de desempenho existentes.** O motivo deste requisito é evitar o desperdício do conhecimento prático já consolidado dos especialistas de desempenho.
- **Lidar com desenvolvimento de software baseado em componentes.** A composição de software é um aspecto estudado no ponto de vista funcional, enquanto que a composição de aspectos não funcionais, como o desempenho, ainda é uma questão em aberto. Fornecer ferramentas para representar este aspecto é uma característica sutil, mas crucial de uma ontologia nesta área. Uma questão em aberto é: “O desempenho de um sistema pode ser modelado a partir do desempenho de seus componentes?”.

Performance-related ontologies and semantic web applications for on-line performance assessment of intelligent systems [29]. A pesquisa relatada em [29] apresenta uma ontologia de desempenho e uma técnica de análise de desempenho para aplicações inteligentes baseadas na *web* semântica. Além disso, o trabalho comenta a construção de regras de desempenho usando OWL para inferência automática de novas restrições de desempenho e conhecimento da Qualidade de Serviço (QoS) do sistema em execução. Assim, esta abordagem pode ser aplicada para detecção de gargalos e garantia da QoS. O metamodelo de desempenho usado neste trabalho é baseado na UML Profile for Schedulability, Performance and Time (SPT). No contexto deste UML *profile*, a ontologia de desempenho pode representar os Service Level Agreements. Algumas das variáveis de desempenho definidas em [29] estão detalhadas a seguir:

- Tempo de resposta: intervalo entre a requisição e a resposta de uma dada transação.

- Vazão (*throughput*): taxa em que requisições são servidas.
- Utilização: probabilidade do serviço ou recurso estar ocupado.
- Demanda: tempo para completar um conjunto composto de requisições.
- Outras variáveis como tempo de serviço, latência, prioridade, capacidade.

Em [29], as variáveis de desempenho são diferenciadas em quatro classes: assumidas (*assumed*), estimadas (*estimated*), obrigatórias (*required*) e medidas (*measured*). Por exemplo, a utilização do processador pode ser monitorada durante a execução e a ontologia deve representar tanto o recurso processador quanto a sua utilização, que é uma medida de desempenho deste recurso. Os recursos podem ser ativos e passivos, por exemplo, um processador é ativo quando estiver executando tarefas ao contrário de sendo acessado por outros passivamente. Desde que esteja ativo, um recurso suporta uma carga (*workload*) representada como a demanda exigida para executar em um cenário específico. Esta carga pode ser gerada por outros objetos do sistema ou vir de fora do modelo. Estas características explicadas anteriormente podem ser visualizadas no modelo ilustrado na Figura 3.

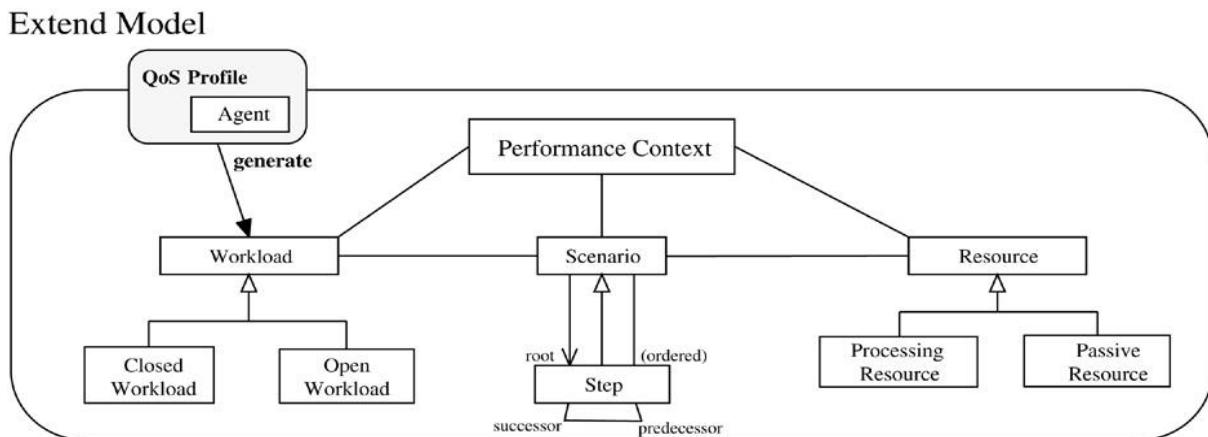


Figura 3: Modelo do domínio de desempenho (Fonte [29]).

Em [29] é utilizada uma descrição em OWL para armazenar, analisar e raciocinar sobre informações relacionadas às métricas de desempenho do sistema em execução. Após implementar a ontologia OWL sobre avaliação de desempenho, foi construída uma aplicação externa (*broker*) para interpretar, computar e atualizar os valores de desempenho dos clientes e componentes do sistema. O raciocínio pode ser baseado em regras definidas pelos usuários em lógica de primeira ordem. A seguir é exemplificado o raciocínio para seleção da tarefa a ser executada de acordo com o *workload* [29].

- **Workload baseado na utilização de recursos:** foram definidas três classes disjuntas denominadas *HardWork*, *MediumWork* e *LightWork*. O conceito *HardWork* foi definido

como qualquer tarefa que utiliza uma quantidade de recurso superior ao seu respectivo limiar (*threshold*), como ilustrado na Figura 4. Portanto, esta classificação depende do ambiente de execução, ou seja, se o recurso for trocado por outro mais poderoso computacionalmente, a classificação pode mudar.

```

WorkLoad(?wor) ^
Resource(?res) ^
hasScenarioWorkLoad(?sce, ?wor) ^
hasScenarioResource(?sce, ?res) ^
utilization(?res, ?pro) ^
swrlb:greaterThan(?pro, ?threshold)
→ HardWork(?wor)

```

Figura 4: Regra em lógica de primeira ordem para definir *HardWork* (Fonte [29]).

- **Workload baseado na prioridade do produtor:** duas classes disjuntas denominadas *HighPriority* e *LowPriorityWork* foram definidas para permitir a classificação do workload com base na prioridade do agente que produz a carga. A Figura 5 ilustra as premissas para inferir que uma instância pertence ao conceito *HighPriorityWork*.

```

Agent(?ag) ^
WorkLoad(?wor) ^
hasProducer(?ag, ?wor) ^
priority(?ag, ?pr) ^
swrlb:greaterThan(?pr, ?threshold)
→ HighPriorityWork(?wor)

```

Figura 5: Regra em lógica de primeira ordem para definir *HighPriorityTask* (Fonte [29]).

- **Workload baseado na eficiência:** para permitir a classificação do workload com base na eficiência foram criados os seguintes conceitos disjuntos: *EfficientExecutionWork* e *NotEfficientExecutionWork*. A Figura 6 exemplifica a lógica que permite deduzir que uma instância é do tipo *EfficientExecutionWork*.

```

LightWork(?x) ^
HighPriorityWork(?x)
→ EfficientExecutionWork(?x)

```

Figura 6: Regra em lógica de primeira ordem para definir *EfficientExecutionWork* (Fonte [29]).

No nível de Qualidade de Serviço (QoS), é possível executar ações ou inferir conhecimento por meio da aplicação de regras. Por exemplo, ações podem parar a execução de uma tarefa que não está de acordo com o nível exigido de QoS. Esta ação é

ilustrada na Figura 7, que considera o tempo de resposta como o parâmetro de avaliação da qualidade. Por outro lado, também é possível detectar recursos que estejam diminuindo a eficiência global da aplicação e reconfigurar o sistema, por exemplo, distribuindo o *workload* para recursos ociosos.

```

WorkLoad(?work) ^ Resource(?re) ^
isExecutedIn(?work, ?re) ^
hasResponseTime(?work, ?Value) ^
hasQoSMaxResponseTime(?work, ?MaxValue) ^
swrlb:greaterThan(?Value, ?MaxValue)
→ Action:StopExecutionWork(?work)

```

Figura 7: Definição de uma Action-Rule para garantia da Qualidade de Serviço (Fonte [29]).

De acordo com os autores [29], a motivação deste trabalho é determinada pela inexistência de técnicas no campo da engenharia ontológica que apoiem uma análise inteligente do desempenho do sistema. Em outras técnicas de desempenho, a maioria das ferramentas usa linguagens próprias para anotar medições ou trocar dados entre ferramentas de modelagem e avaliação. O reuso de medições é feito por relacionamentos estruturais (XML *parsers* ou bancos de dados relacionais), contudo a semântica não é considerada. Além disso, existem novas áreas onde a riqueza semântica das informações pode auxiliar no processo de tomada de decisão [29]. Sistemas inteligentes requerem um processo de decisão baseado na avaliação de desempenho e na informação da Qualidade de Serviço. Portanto, o modelo deve levar em consideração características dos serviços como prioridades, limiares, etc. Seguindo o exemplo do processador, uma maneira natural de expressar sua Qualidade de Serviço poderia ser: “o processador não deve exceder 90% de utilização durante a execução, se isto acontecer, então ...” e o sistema automaticamente dispara alguma regra pré-definida para garantir o Service Level Agreement sobre a utilização do processador. Este artigo [29] mostra que é possível raciocinar sobre o desempenho em um ambiente inteligente e até mesmo executar ações para influenciá-lo durante a execução. Assim, segundo os autores, o conhecimento acumulado na Engenharia de Desempenho de Software deve incluir novas questões relacionadas a ontologias.

Web Performance and Behavior Ontology [11]. Este trabalho [11] define uma arquitetura para melhoramento contínuo do desempenho de sistemas *web* durante sua execução. Para isso, são usadas informações sobre o contexto atual do sistema, o que

inclui os usuários, os servidores e as medições do desempenho ao longo do tempo. É importante lembrar que a referência [29] também apresenta estudos sobre a adaptação dinâmica de sistemas de acordo com o estado atual e o ambiente. Em [11] é proposta uma abordagem que utiliza ontologias para representar as seguintes informações:

- 1) **Behavior Knowledge Base:** módulo centralizado que armazena em ontologias informações sobre os elementos significativos de um sistema *web* (usuários, requisições, *proxies*, nodos de cache, *gateways*, servidores, etc).
- 2) **Web System Elements Knowledge Base:** base que mede, representa e analisa informações sobre o desempenho de cada elemento contido no sistema de interesse. Para cada parte do sistema, devido as suas características próprias, deve-se usar uma descrição ontológica. Além disso, a Base de Conhecimento Comportamental (Behavior Knowledge Base) deve possuir acesso às informações armazenadas nesta base.

Por meio de inferências na Base de Conhecimento Comportamental, é possível modificar o estado global do sistema de modo dinâmico. O desempenho de um sistema é uma característica resultante da combinação do desempenho de cada uma de suas partes, o que pode incluir camadas de banco de dados, servidor *web*, etc. Portanto, para melhorar o desempenho global do sistema é preciso considerar informações de todas as suas camadas. Este trabalho [11] parte do princípio de que o comportamento de cada elemento do sistema seja modelado em uma ontologia e armazenado em uma base de conhecimentos. Estas ontologias possuem conhecimento sobre o funcionamento do sistema, como por exemplo, informações sobre o balanceamento de carga, políticas de cache, priorização de tarefas, etc. Deste modo, os *reasoners* usam estas informações para aperfeiçoamento (*tuning*) do sistema, o que é feito por meio da aplicação de regras, heurísticas e operações. Quando o estado do sistema é alterado, o *reasoner* avalia o comportamento e o desempenho dos elementos neste novo estado. Depois de um período de tempo, se o desempenho melhorou, o *reasoner* considera este novo estado como estável (*stable*). Porém, se o desempenho piorou, o *reasoner* restaura o estado antigo e tenta alterações diferentes no sistema. Para criar um novo estado, o *reasoner* aplica operações selecionadas de acordo com parâmetros da Base de Conhecimentos Comportamental. Esta abordagem é ilustrada na Figura 8.

Em [11], é acrescentada uma camada de cache entre os clientes e o servidor para armazenar as requisições geradas, reduzindo o custo computacional e o tempo de resposta. Ao contrário de armazenar páginas inteiras, são armazenados fragmentos de páginas na tentativa de aumentar o desempenho. A justificativa para isto é que, a melhor

possibilidade de fragmentação, do ponto de vista do desempenho, pode mudar ao longo do tempo. Portanto, para inferir a melhor forma de fragmentação, a camada de cache coleta informações sobre:

- Usuários: páginas mais solicitadas, caminhos de navegação, etc.
- Servidor: tempo de resposta e tamanho por requisição, taxa de mudança no conteúdo de cada requisição e taxa de compartilhamento entre conteúdos de páginas *web*.

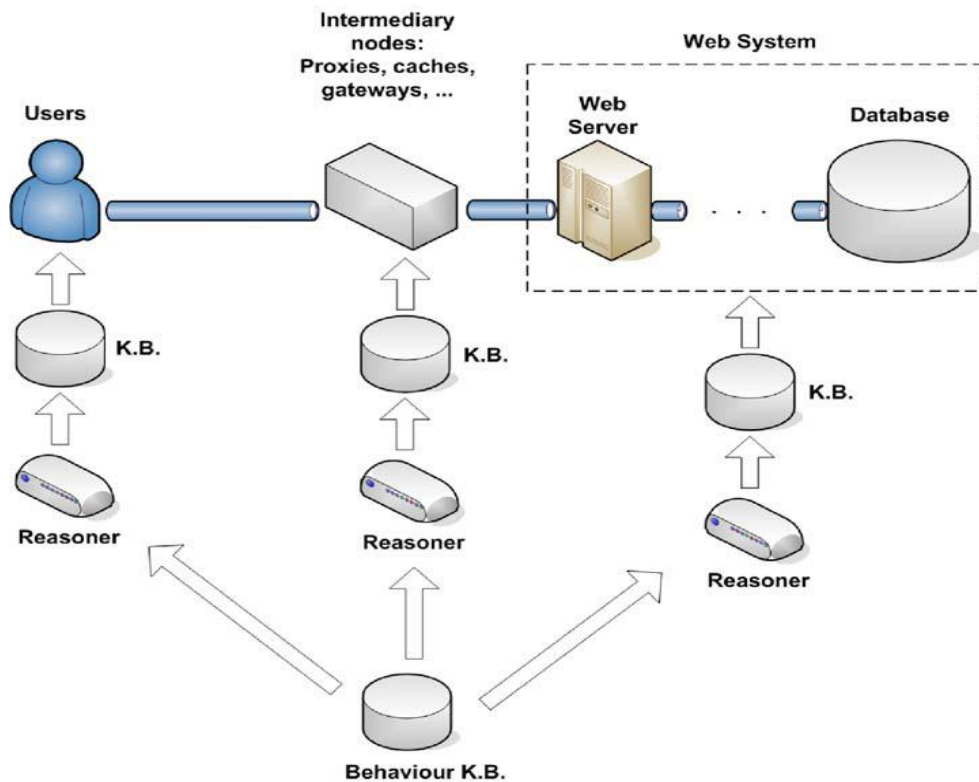


Figura 8: Abordagem dinâmica de aperfeiçoamento do desempenho (Fonte [11]).

Os autores de [11] consideram a ampliação deste trabalho em dois ramos diferentes. O primeiro é a criação de linguagens ontológicas específicas para modelar os diferentes elementos de um sistema *web* e a integração dessas ontologias nos elementos que usariam a informação armazenada na base de conhecimento correspondente. O segundo ramo é a definição de operadores e regras para alterar os modelos de cada elemento de um sistema *web* usando a informação da base de conhecimento de desempenho.

An Approach to Ontology-aided Performance Engineering through NFR Framework [56]. Este trabalho [56] teve como objetivo formalizar uma base de conhecimento que incluísse medidas, como tempo de resposta, e técnicas úteis para atingir um bom desempenho, como indexação. Segundo os autores, este conhecimento deve ser representado de uma forma intuitiva, fácil de ler, apoiado por diagramas e, se

possível, em uma linguagem formal tanto sintática quanto semanticamente. A pesquisa descrita em [56] se baseia no Non-Functional Requirement (NFR) Framework Softgoal Interdependency Graph. Este framework auxilia os desenvolvedores a lidarem com requisitos não funcionais, o que inclui, entre outros, o desempenho. Um softgoal define características desejáveis do sistema, eventualmente qualitativas e podendo não apresentar critérios de satisfação bem definidos. A Figura 9 ilustra um exemplo de Grafo de Interdependência de Softgoals, onde as nuvens representam softgoals e as flechas conectando as nuvens representam Interdependências entre softgoals. Se as Interdependências estiverem rotuladas com o símbolo +, então significa que a contribuição é positiva. Se a nuvem estiver em negrito, então o softgoal é do tipo operacionalização, que representa técnicas para atingir outros softgoals. Os símbolos ✓ e ✗ dentro das nuvens foram colocados por um procedimento que avalia os softgoals de baixo para cima, seguindo as interdependências. A Figura 9 ilustra o modelo proposto e uma explicação mais detalhada pode ser encontrada em [40].

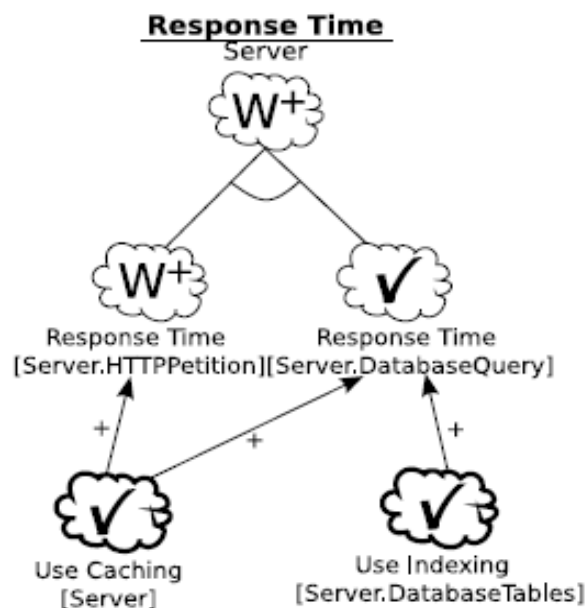


Figura 9: Exemplo de um Grafo de Interdependência de Softgoals (Fonte [56]).

Em [56] são apresentadas duas ontologias projetadas em OWL: uma para descrever o Grafo de Interdependência de Softgoal e outra para representar o tipo dos requisitos não funcionais (NFR). Assim, é possível usar uma notação mais formal que adiciona semântica e capacidade de inferência por computadores. Uma das ontologias proposta contém o metamodelo da descrição hierárquica dos tipos de NFR. Esta primeira ontologia é estruturada em formato de árvore e possui apenas uma classe denominada *NFR_Type*

com duas propriedades: *hasSubtype* e *isSubtypeOf*. Estas propriedades indicam a hierarquia (ascendente e descendente) dos requisitos não funcionais. A segunda ontologia descreve os Grafos de Interdependência de Softgoals (SIG), que são representados como instâncias das classes. Esta segunda ontologia possui classes para representar os *Softgoals* e as Interdependências. Para definir uma Interdependência é preciso relacionar qual o seu tipo e também qual a sua contribuição. O *type* da ontologia anterior, que foi importada nesta, é referenciado e combinado com um atributo *String* denominado *topic*. Dessa forma, é possível representar que atingir um bom *throughput* (*type*) para uma impressora (*topic*) não é o mesmo que para um banco de dados. Segundo os autores [56], a aplicação destas ontologias pode auxiliar no processo de tomada de decisão baseado no conhecimento de decisões anteriores que impactaram negativa ou positivamente no desempenho. E se a experiência acumulada de todos os projetos de software estivesse disponível? Os softwares *designers* poderiam escolher soluções que funcionaram bem para outros e evitar escolhas prejudiciais. Os autores deste artigo denominaram esta base de conhecimento de Software Performance Engineering Body of Knowledge (SPEBoK). Nesta base, poderia ser armazenado tudo o que se sabe sobre a Engenharia de Desempenho, o que inclui qualquer abordagem comprovadamente útil ou prejudicial do ponto de vista do desempenho. Contudo, o conhecimento armazenado não precisa ser restrito às questões de desempenho, por exemplo, a criptografia de dados aumenta a segurança, mas pode degradar o tempo de resposta. Este artigo mostra os primeiros blocos de construção do SPEBoK e a intenção dos autores [56] é combinar a Engenharia de Software e a Engenharia de Desempenho para permitir a troca de conhecimentos e a definição de um vocabulário comum.

Using ontologies to improve performance in a web system – A web caching system case of study [10]. Em [10], dados sobre o estado do ambiente e o desempenho do sistema são armazenados em ontologias e usados para otimizar o desempenho por meio da configuração do sistema em tempo de execução. A proposta descrita em [10] é semelhante à abordagem explicada em [11], onde cada elemento do sistema possui suas características próprias e isto é modelado por meio de ontologias. Em [10] são implementadas funções de monitoramento. Segundo os autores, o objetivo principal a ser alcançado no futuro é uma arquitetura completa de monitoramento e atuação, onde atuadores são adicionados para modificar o sistema. Em outras palavras, os atuadores fazem uso de regras de inferência para analisar os dados monitorados e tomar decisões sobre a configuração do sistema.

3.2. Ontologias e Teste de Software

“The insight at the root of artificial intelligence was that these bits (manipulated by computers) could just as well stand as symbols for concepts that the machine would combine by the strict rules of logic or the looser associations of psychology.”
(Daniel Crevier)

A seção anterior apresentou trabalhos que abordaram simultaneamente os temas ontologia e desempenho de software. Esta seção é responsável por apresentar trabalhos que relacionem ontologias no domínio de teste de software. É esperado que uma ontologia de teste forneça um vocabulário para representar e comunicar o conhecimento sobre o domínio de teste de software. Além disso, tal ontologia deve apresentar um conjunto de relacionamentos existentes entre os conceitos do domínio [39]. Com o objetivo de entender melhor como essa questão é tratada na literatura, a seguir são apresentadas sínteses de trabalhos sobre ontologias em teste de software.

Ontology-Based Web Application Testing [61]. De acordo com a referência [61], para automatizar a geração e a execução de casos de testes usando ontologias são obrigatórias duas etapas. A primeira etapa consiste em desenvolver uma ontologia que capture em um nível adequado o conhecimento necessário para efetuar o processo de teste. O conhecimento sobre o processo de teste inclui os diferentes tipos de teste, seus objetivos, limitações, capacidades e atividades envolvidas. Esta ontologia de teste de software poderia ser criada com base no *SWEBOK* e reusada em qualquer novo teste a ser executado. Além disso, é necessário codificar também, em formato processável por máquina, o conhecimento sobre o domínio da aplicação a ser testada. Este conhecimento inclui os conceitos, possibilidades, limitações, relacionamentos e funcionalidades esperadas da aplicação sob teste. A ontologia do domínio da aplicação pode ser elaborada simultaneamente com o desenvolvimento da aplicação em si. A etapa final consiste em desenvolver procedimentos que usem o conhecimento embarcado nestas ontologias para automatizar as tarefas de teste. Em [61] são citados 5 exemplos teóricos para ilustrar como ontologias podem auxiliar na automação de testes em aplicações *web*:

1) Especificação e planejamento do teste: usando uma ontologia que contém o conhecimento das atividades de teste (como ordem e relacionamentos), é possível especificar o plano de teste de forma automática. Por exemplo: se a especificação diz que “o sistema x deve ser testado usando uma estratégia de caixa preta”, pode ser inferido

que tipos de testes e em qual ordem devem ser executados. Ainda é possível definir quais critérios de teste e quais métodos de geração de casos de teste devem ser usados.

2) Consulta semântica: o uso de ontologias nas atividades de teste (como planejamento, especificação, execução e análise) permite a geração automática de documentos do processo inteiro. Segundo [61], seria possível recuperar informações do processo de teste usando consultas semânticas, como, por exemplo, após executar um teste de cobertura do código na aplicação, seria possível perguntar quais classes ou métodos não foram suficientemente testadas.

3) Usar ontologias como facilitadoras: ontologias podem auxiliar na definição, publicação, registro, anúncio e recuperação de *web services*. Os *web services* (ou também agentes de software) podem ser usados nas diferentes atividades do processo de teste e as ontologias podem ser usadas como meio de comunicação, compartilhando o conhecimento do domínio e aumentando a cooperação entre os agentes ou serviços.

4) Geração dos casos de teste: se a geração for baseada na ontologia de teste de software, o tipo de teste a ser executado define o método de geração. Por exemplo: ao executar um teste de segurança em aplicações *web*, pode ser usado *SQL injection* para gerar os dados que vão preencher os campos do formulário. Contudo, se a geração for baseada na ontologia de domínio da aplicação, é possível gerar dados semanticamente válidos para preencher as requisições.

5) Oráculos de teste: um oráculo de teste julga o resultado dos testes para decidir se passaram ou falharam. Este julgamento é baseado em critérios que podem ser definidos formalmente em ontologias. Por exemplo, quando executando um teste de desempenho em aplicações *web*, os resultados podem ser julgados com base no atraso da resposta. Ou, igualmente, os resultados podem ser julgados com base no código de status das respostas HTTP ou se um registro foi inserido ou alterado de um banco de dados.

Towards the Establishment of an Ontology of Software Testing [19]. Em [19] é apresentada a *OntoTest*, uma ontologia de teste de software que auxilia na aquisição, organização, reuso e compartilhamento de conhecimentos sobre a área. A *OntoTest* explora os diferentes aspectos envolvidos na atividade de teste de software, define um vocabulário comum e também auxilia no estabelecimento de arquiteturas de referência. Tais arquiteturas são responsáveis por facilitar o uso e a integração de ferramentas, processos e artefatos em ambientes de engenharia de software [19]. Neste sentido, a *OntoTest* foi usada para estabelecer uma arquitetura de referência sobre teste de software, denominada *RefTEST*. Para estabelecer arquiteturas de referência é preciso

conhecimento profundo sobre o domínio, mas uma vez estabelecidas, o conhecimento do domínio é associado em suas atividades e relacionamentos. Outro cenário possível de aplicação desta ontologia é o estabelecimento de um processo de aprendizagem sobre teste de software. Assim, a ontologia pode facilitar o acompanhamento de mudanças no conhecimento de teste, assim como o ensinamento deste novo conhecimento por meio de uma abordagem sistemática [19]. A Figura 10 ilustra a estrutura da OntoTest:

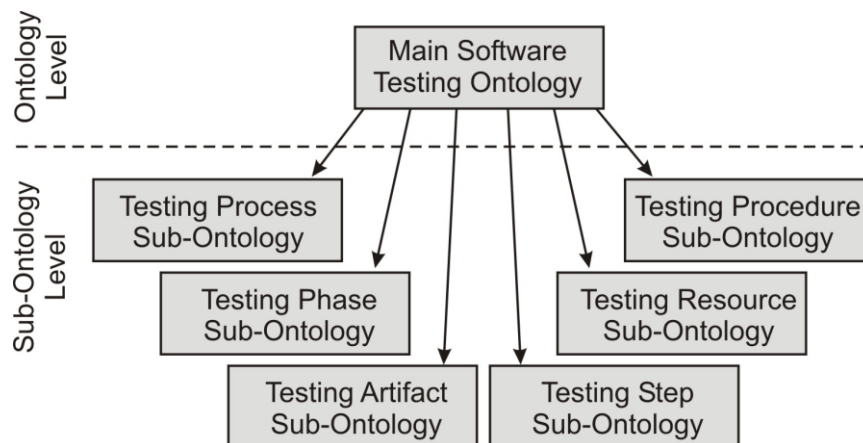


Figura 10: Estrutura da OntoTest (Fonte [19]).

Knowledge-based Software Test Generation [71]. Este trabalho [71] aponta que a geração de testes deve considerar três aspectos:

1) A especificação do que deve ser testado, que geralmente é definida em oráculos de teste e critérios de cobertura. A especificação define os requisitos e o correto comportamento do software. A abordagem proposta em [71] consiste na utilização de ontologias e regras para descrever este conhecimento. Esta ontologia é então combinada com uma ontologia do modelo mental do especialista em testes, que pode incluir conhecimentos sobre a implementação, aspectos propensos a erros e outros. Por fim, os critérios de cobertura são especificados de acordo com um padrão ou pelo especialista.

2) A identificação dos objetivos do teste, os quais, nesta abordagem proposta, são deduzidos sobre a especificação representada no oráculo de teste e nos critérios de cobertura. Cada objetivo de teste representa um caso de teste, porém podem existir objetivos redundantes, devendo ser aplicadas regras de verificação de redundância.

3) A geração de casos de teste de acordo com os objetivos identificados. Técnicas de verificação de modelos, planejamento de IA ou algoritmos de travessia de grafos são usadas para gerar os casos de teste, que são escritos em uma ontologia de suíte de testes independente de linguagem de programação. Os casos de testes executáveis são

gerados com base nesta ontologia e em outra que representa o conhecimento de implementação dependente de uma linguagem de programação. Segundo os autores [71], esta última ontologia contém informações sobre o código do programa a ser testado e pode ser gerada por engenharia reversa.

A abordagem proposta baseada em ontologias desacopla estes três aspectos relativos à geração de testes, e, com isso, aumenta o nível de abstração da representação do conhecimento [71]. Além disso, se especialistas em testes ampliarem o oráculo de teste com critérios de cobertura personalizados e conhecimentos sobre aspectos do sistema propensos a erros, a qualidade dos testes gerados tende a aumentar. A arquitetura proposta em [71] é abstrata, como ilustrada na Figura 11, e pode ser construída com tecnologias variadas. Os autores implementaram essa abordagem para a geração de casos de teste unitário baseado em máquinas de estados e utilizaram as tecnologias JUnit, OWL, POSL e OO jDREW. Os autores citam que um próximo passo seria expandir esta técnica para as fases de teste de integração e de sistema [71].

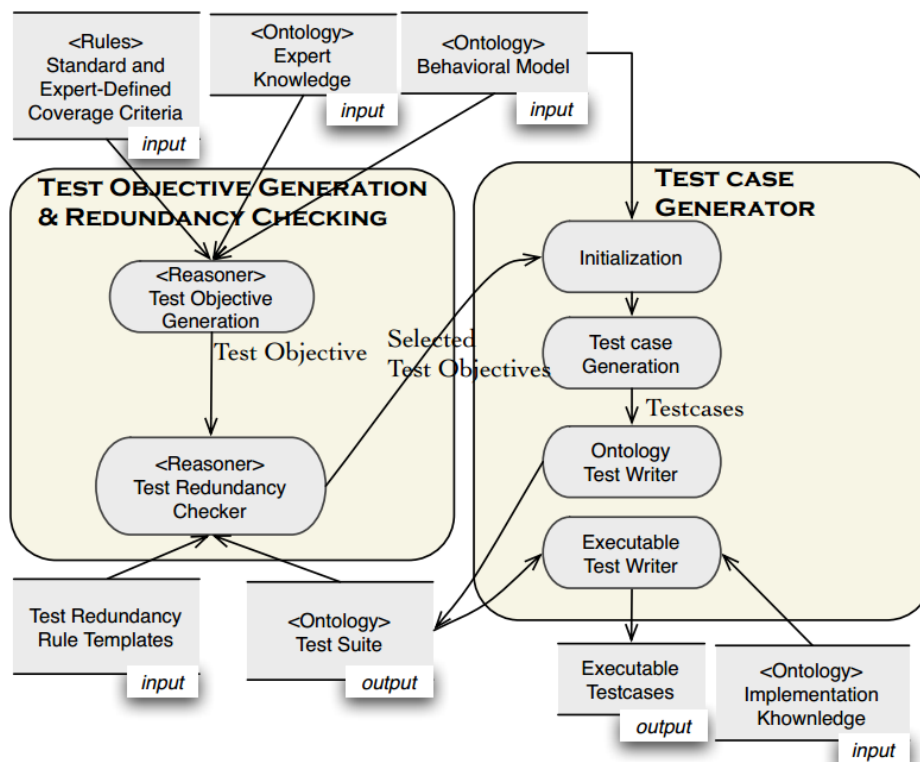


Figura 11: Abordagem baseada em ontologias para geração de casos de teste unitário (Fonte [71]).

Test Case Reuse Based on Ontology [39]. Este trabalho [39] explica um método responsável por pesquisar e recuperar casos de testes similares de acordo com a informação semântica representada em uma ontologia. Os autores [39] exemplificam que

conceitos como “automação de teste de desempenho” e “automação de teste funcional” são similares, mas possuem menos similaridade do que “teste de desempenho” e “LoadRunner”. Um cálculo de distância semântica é usado para a inferência destas similaridades. O processo de reuso de casos de teste baseado em ontologias inclui:

- 1) Construção da ontologia de teste de software usando o Protégé com a ajuda de especialistas do domínio.
- 2) Coleta do conjunto de casos de teste e anotação de cada caso de teste de acordo com a ontologia estabelecida. Os casos de testes e seus metadados originados por esta anotação são armazenados em uma base de dados.
- 3) Pré-processamento da consulta do usuário (de acordo com a ontologia definida).
- 4) Inferência pelo módulo *reasoner*, de acordo com a ontologia, e tradução da consulta em casos de testes a serem recuperados.
- 5) Devolução do resultado encontrado na interface de usuário.

Research and Implementation of Knowledge Management Methods in Software Testing Process [41]. Em [41] é apresentado um sistema para gerenciamento do conhecimento de teste de software por meio da recuperação de documentos e identificação de especialistas. Para isto, ontologias são utilizadas para identificar os seguintes conceitos: documentos, referências, projetos, quadro de funcionários (*staff*) e nível de conhecimento. O sistema proposto trabalha com uma base de dados de documentos, onde os usuários podem adicionar, recuperar e avaliar documentos usando diversos critérios. Analistas de conhecimento podem alterar o nível de conhecimento dos usuários de acordo com uma análise de seus históricos. A recuperação do conhecimento consiste na identificação de um documento que atenda as necessidades do usuário. Caso não exista, o sistema pode indicar um especialista capaz de resolver o problema. A ontologia representa o papel de base de conhecimento no sistema de recuperação de informação, classificando os conceitos, relacionamentos, atributos, instâncias e restrições nos conceitos sobre teste de software. Assim, é possível identificar conceitos ou atributos correlacionados ao recuperar o conhecimento requisitado pelo usuário [41].

Ontology-based Web Service Robustness Test Generation [42]. Este trabalho gera testes de robustez utilizando uma ontologia do domínio dos serviços *web* que serão testados. A propriedade “robustez” é definida pela IEEE [28] como o grau em um sistema ou componente pode funcionar corretamente mesmo na presença de entradas inválidas ou condições ambientais estressantes. A ontologia especifica semanticamente os serviços e *workflows*, e com isso é possível derivar restrições sobre as classes, propriedades e

dependências dos parâmetros. De acordo com essas restrições, são aplicados operadores de mutação de dados nos casos de testes funcionais para gerar dados de teste de robustez. Segundo os autores de [42], anteriormente a este estudo, somente era possível verificar violações sintáticas e de *workflow* nos serviços, mas com o uso da ontologia é possível violar restrições semânticas. A Figura 12 ilustra as três abordagens adotadas na geração de teste de robustez, dando ênfase a parte tracejada em vermelho que corresponde à abordagem proposta em [42].

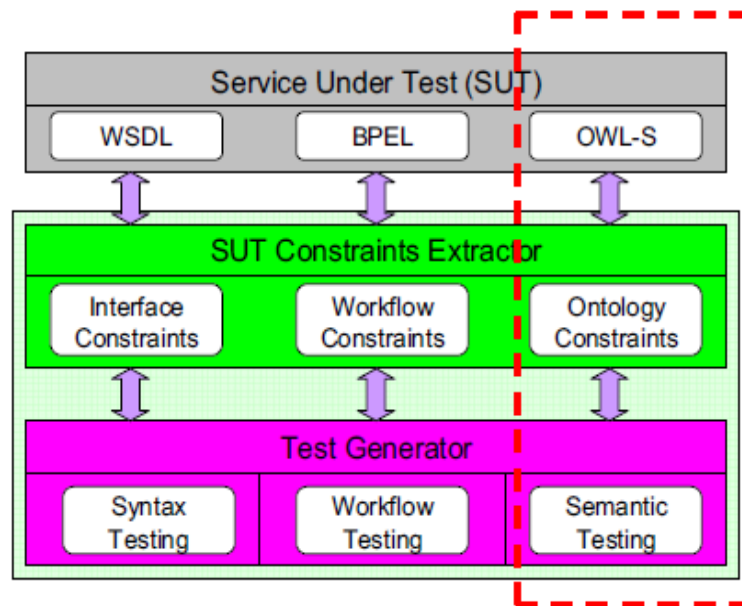


Figura 12: Robustness Testing Framework (Fonte [42]).

An Ontology-based Software Test Generation Framework [70]. Em [70] é definido um framework para geração automática de suítes de teste baseado em ontologias. Esta geração foi dividida em quatro etapas, como ilustrado na Figura 13 e explicado a seguir:

Fase 1) Geração dos objetivos do teste. Para executar esta etapa é preciso definir uma representação ontológica do que deve ser testado, o que inclui a especificação do modelo comportamental, o conhecimento de especialistas e os critérios de cobertura. A saída desta etapa é um conjunto de objetivos do teste. Para isso, são executadas inferências nas ontologias que descrevem a especificação.

Fase 2) Verificação de redundância. Para cada objetivo de teste, uma regra de verificação de redundância é usada para conferir se o objetivo do teste é satisfeito por um caso de teste já incluído na ontologia de testes abstratos. Esta fase também utiliza inferência nas ontologias. Esta etapa é executada em conjunto com a Fase 3.

Fase 3) Geração da ontologia de testes abstratos. Para cada objetivo não redundante

de teste, um caso de teste abstrato é gerado e adicionado na parcialmente gerada ontologia de testes abstratos. Esta fase é executada usando métodos de geração de testes predominantes na literatura (como *graph traversal algorithms*).

Fase 4) Geração da suíte de teste executável. Para cada caso de teste abstrato da ontologia de testes abstratos, um caso de teste executável é gerado usando a ontologia de conhecimento da implementação.

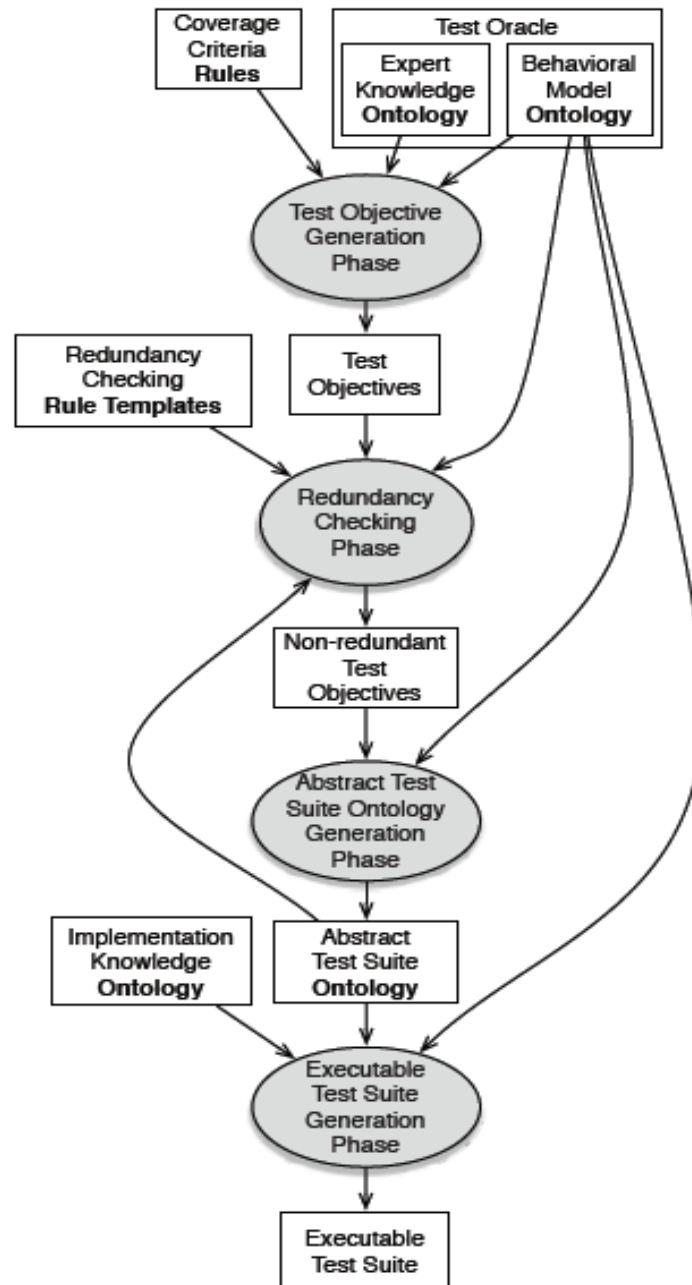


Figura 13: Visão geral do framework para geração de teste baseado em ontologias (Fonte [70]).

Ontology-Based Test Case Generation for Simulating Complex Production Automation Systems [67]. Este trabalho [67] apresenta uma abordagem de extração do

conhecimento de especialistas de teste. O conhecimento é representado explicitamente em uma ontologia, que é usada para gerar casos de testes para um simulador de linha de montagem automatizada. Os autores [67] argumentam que a abordagem apoiada por ontologias não exige conhecimento de programação e apresenta maior facilidade de alteração dos casos de teste, porém é inicialmente mais complexa de aplicar. A ontologia deve modelar tanto o conhecimento sobre o domínio de teste, quanto o conhecimento sobre o sistema a ser testado. Embora os usuários tenham que modificar a ontologia com ferramentas de edição gráfica (como o Protégé), esta tarefa envolve menos esforço do que modificar o código *hardcoded* de um *script* [67]. O processo proposto de geração de casos de teste é dividido em três fases. A primeira fase carrega a ontologia que é usada para gerar dinamicamente uma interface gráfica de acordo com os parâmetros de teste. A segunda fase corresponde à parametrização dos testes na interface gráfica gerada. Na terceira fase, os casos de teste são gerados e exportados para arquivos XML.

Ontology-based Test Generation for MultiAgent Systems [12]. A pesquisa feita em [12] utiliza ontologias de interação entre agentes para definir o conteúdo semântico da comunicação entre os agentes. A finalidade deste estudo é automatizar a geração de teste por sistemas multiagentes. De acordo com a ontologia é possível verificar as mensagens trocadas e também gerar entradas válidas ou inválidas para testar o comportamento dos agentes [12]. Esta abordagem foi integrada em um *framework* de teste denominado *eCat*, o qual gera e executa casos de teste automaticamente. Para gerar os testes, é feito um alinhamento de uma ontologia que caracteriza o domínio dos agentes com a ontologia de interação. Este trabalho avalia também a exploração do espaço de entradas e a possibilidade de usar ontologia como oráculo de teste. Se as entradas do teste forem inválidas, o objetivo é avaliar a robustez do sistema (o que também foi feito usando ontologias em outro trabalho [42]). A abordagem adotada no *eCat* é a seguinte: o *Tester Agent* seleciona o *template* de um caso de teste, invoca o *O-based Generator* para preencher os valores e executa o teste. Enquanto isto, os *Monitoring Agents* observam o comportamento dos agentes e reportam se alguma falha ocorreu. Além disso, o *Tester Agent* verifica se as mensagens recebidas estão de acordo com a ontologia ou não. Este ciclo de geração e execução de casos de testes pode continuar e avançar para a próxima iteração até que seja interrompido. Os autores definem que os casos de teste, que são codificados em XML, podem ser uma sequência de interações definidas pelo usuário ou podem seguir um protocolo padrão de interações como o FIPA (The Foundation for Intelligent Physical Agents) [12].

Developing a software testing ontology in UML for a software growth environment of web-based applications [26]. A ontologia de teste de software proposta neste trabalho [26] representa os seguintes conceitos: *tester*, *context*, *method*, *artifact*, *activity* e *environment*. Esta ontologia possui semelhanças com a OntoTest, comentada anteriormente e proposta em [19]. O *tester* define o ser humano ou software que possui determinada capacidade de executar uma atividade de teste. O *context* se refere à etapa ou ao objetivo de realização da atividade de teste, por exemplo: teste unitário, de integração, de sistema ou outros tipos. É o contexto que determina o método de teste mais adequado, assim como as entradas e saídas da atividade de teste. O conceito *activity* inclui atividades de planejamento, geração de casos de teste, execução e validação dos resultados. O termo *method* define métodos aplicáveis nas atividades de teste e que podem variar quanto às técnicas e abordagens utilizadas. O conceito *artifact* possui as propriedades tipo (plano de teste, script de teste, resultado, etc) e formato (arquivo HTML, PDF, etc). Os artefatos também possuem uma localização, um conteúdo, um histórico e podem estar envolvidos nas atividades de teste. Por fim, o *environment* caracteriza as configurações do ambiente de hardware e software de execução do teste. Usando estes conceitos básicos definidos anteriormente, em [26], os autores estabelecem conceitos compostos como capacidade e tarefa. A capacidade de um *tester* é determinada pela atividade que ele pode executar, de acordo com um determinado contexto, método, ambiente e artefatos de entrada e saída. Uma tarefa consiste de uma atividade de teste e informações relacionadas sobre como a atividade deve ser executada (contexto, método, ambiente e artefatos de entrada e saída). Usando estes conceitos, os autores definem as seguintes relações parciais de ordem: subsunção (*subsumption*) entre métodos de teste, compatibilidade entre formatos de artefatos, melhoramento entre ambientes, inclusão entre atividades e sequenciamento temporal de atividades de teste. Com estas relações, é possível inferir que um *tester* possui capacidade igual ou superior a outro, pois consegue executar todas as tarefas que o outro pode fazer. Da mesma forma, pode ser deduzida a relação que uma tarefa contém outra(s) e a relação de combinar uma tarefa com o *tester* de maior capacidade para executá-la. A referência [57], destes mesmos autores, utiliza esta ontologia como meio de comunicação de agentes que possuem o objetivo de testar aplicações *web*.

A multi-agent software environment for testing web-based applications [57]. Este trabalho [57] utiliza a ontologia de teste de software descrita anteriormente (em [26]) para definir o conteúdo das mensagens trocadas entre agentes de software que possuem

o objetivo de testar aplicações *web*. Neste caso, a ontologia funciona como facilitadora, pois é usada para apoiar a comunicação entre os agentes. Nesta abordagem, agentes intermediários usam a ontologia como forma de inferência para gerenciar o conhecimento sobre os agentes e atribuir cada tarefa para o agente mais apropriado. Os conceitos principais da ontologia são: *method*, *artifact*, *testing context*, *tester*, *activity* e *environment*. O uso desta ontologia permite uma integração flexível dos múltiplos agentes e a implementação de diversos métodos diferentes de teste [57]. Além disso, o agente mais adequado para executar novas tarefas em um determinado contexto e momento no tempo é escolhido de forma dinâmica.

Ontology-Based Test Case Generation for Testing Web Services [74]. O trabalho descrito em [74] combina ontologias que descrevem *web services* (OWL-S) e redes de Petri para gerar casos de teste automaticamente. Para cada serviço, são analisados os parâmetros de entrada necessários, a saída gerada, pré-condições e pós-condições de sua execução. Com base na rede de Petri, processos de teste são gerados para cobrir diversos caminhos possíveis de execução. Os dados de teste são gerados por meio de inferências na ontologia. Uma visão geral da abordagem baseada em ontologias para geração de casos de teste proposta por estes autores pode ser visualizada na Figura 14.

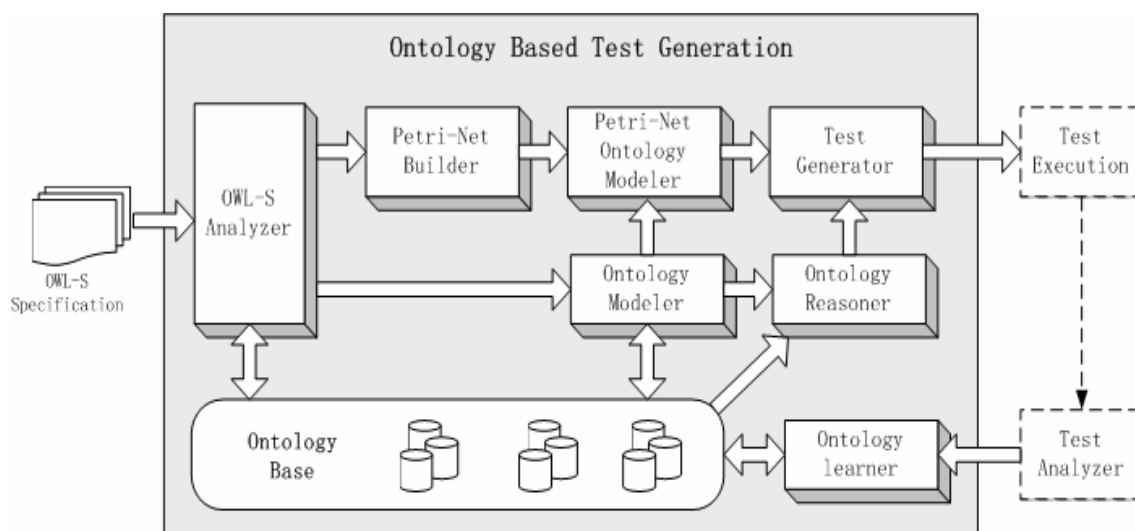


Figura 14: Abordagem baseada em ontologias para geração de testes para *web services* (Fonte [74]).

Esta seção analisou trabalhos relacionados que aplicaram ontologias relacionadas ao teste de desempenho de software. A seção seguinte da dissertação apresentará a ontologia proposta neste trabalho de mestrado.

4. ONTOLOGIA PROPOSTA

"Computers are useless. They can only give you answers." (Pablo Picasso)

A ontologia proposta sobre teste de desempenho foi construída com o apoio do software Protégé [62] e está representada na linguagem OWL [59]. OWL é uma linguagem destinada a representar explicitamente o significado de termos em vocabulários e as relações entre estes termos [59]. Além de ser caracterizada por uma semântica formal, a linguagem OWL foi projetada com a finalidade de permitir uma forma comum de processar informações. A metodologia de desenvolvimento de ontologia adotada foi a proposta por Noy e McGuinness [49]. Visto que a primeira etapa da metodologia seguida consiste em determinar o escopo da ontologia, este é o tema abordado na próxima seção.

4.1. Escopo da Ontologia

A ontologia possui como domínio o teste de desempenho, portanto um dos seus objetivos consiste em formalizar os principais conceitos, relacionamentos e indivíduos deste domínio. Além disso, a ontologia proposta tem como objetivo auxiliar a gerência dos testes de desempenho de software, possuindo como usuários finais os testadores de desempenho. Uma forma de determinar o escopo da ontologia é esboçar uma lista de questões que devem ser respondidas por uma base de conhecimento que utilize-a. Estas perguntas são chamadas questões de competência [49], sendo responsáveis por facilitar o delineamento do escopo da ontologia. Portanto, a seguir estão listadas questões de competência que a ontologia proposta deve responder:

- Quais características devem ser consideradas durante a elaboração de um teste de desempenho? Em outras palavras, quais os principais conceitos e propriedades sobre o domínio de teste de desempenho que um testador deve ter conhecimento?
- Quais tecnologias podem ser utilizadas em uma determinada configuração de um sistema que se deseja testar o desempenho?
- Quais atividades o teste de desempenho deve possuir para atingir o objetivo do teste?
- Dado uma ferramenta de teste, quais métricas podem ser coletadas e quais protocolos podem ser testados?
- Quais ferramentas podem ser utilizadas para gerar carga ou monitorar um determinado

ambiente de teste? Por exemplo, quais ferramentas para teste de desempenho podem ser utilizadas para monitorar o sistema operacional Linux?

Representando o conhecimento sobre o domínio teste de desempenho em uma ontologia, as respostas para estas perguntas podem ser deduzidas com base nos conceitos, propriedades, instâncias e axiomas. Para responder tais perguntas é necessário representar informações como as restrições e as capacidades envolvidas na aplicação de cada tecnologia, *e.g.*, uma capacidade fornecida pela ferramenta de teste de desempenho LoadRunner é conseguir testar aplicações sobre o protocolo HTTP. Dessa forma, uma vez definido o ambiente de teste, é possível escolher quais tecnologias podem ser utilizadas em uma determinada instância do conceito teste de desempenho. Além disso, de acordo com as escolhas na elaboração de um teste de desempenho, um plano de teste pode ser gerado por uma aplicação que utilize tal ontologia. O plano de teste é um documento que fornece a base para a execução dos testes de software de modo organizado, sistemático e bem documentado [27]. Estas questões de competência foram mapeadas nos casos de uso ilustrados na Figura 15. Em um nível alto de abstração o objetivo da aplicação baseada na ontologia é auxiliar os testadores a definir o que deve ser levado em consideração para a execução de um teste de desempenho. Assim, apoiando as decisões dos testadores como, por exemplo, sugerindo uma metodologia para elaboração dos testes ou indicando ferramentas alinhadas aos objetivos de teste.

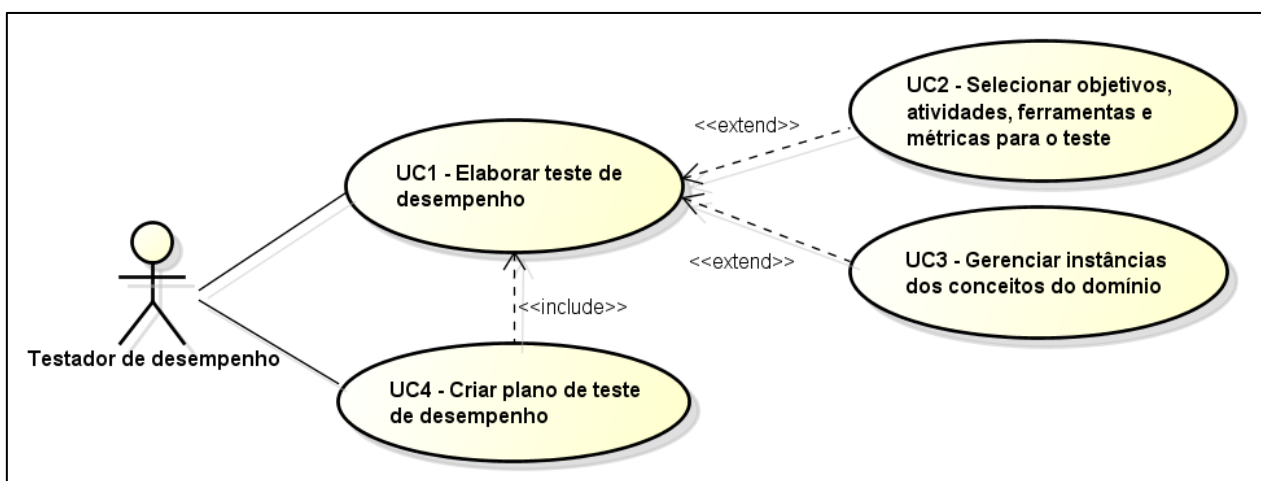


Figura 15: Diagrama de casos de uso descrevendo o escopo de aplicações baseadas na ontologia proposta.

UC1 – Elaborar teste de desempenho. Quais características um teste de desempenho pode possuir? Como selecionar tecnologias e ferramentas compatíveis com o ambiente de teste? O caso de uso “Elaborar teste de desempenho” inclui a criação de

uma instância do conceito que representa o teste de desempenho na ontologia. Esta nova instância deve obedecer as restrições deste conceito, podendo possuir relacionamentos com outros conceitos de acordo com as propriedades existentes na ontologia. Por exemplo, um teste de desempenho pode se relacionar com conceitos como os objetivos, as atividades, as ferramentas e as métricas de desempenho. Desta forma, a ontologia deverá representar os objetivos que um teste de desempenho pode possuir, as atividades executadas nos testes de desempenho, as ferramentas que podem resolver tais tarefas e as métricas de desempenho que podem ser monitoradas. Como ilustrado na Figura 15, o caso de uso “Elaborar teste de desempenho” pode ser estendido pelos casos de uso “Selecionar objetivos, atividades, ferramentas e métricas para o teste” e “Gerenciar instâncias dos conceitos do domínio”.

UC2 – Selecionar objetivos, atividades, ferramentas e métricas para o teste.

Este caso de uso permite que o testador selecione instâncias da ontologia que não possuam relacionamentos com um determinado indivíduo do conceito teste de desempenho. Em outras palavras, quais objetivos, atividades, ferramentas e métricas estarão relacionadas com um teste de desempenho. É preciso definir também qual o ambiente de teste, a aplicação a ser testada e quais protocolos a aplicação utiliza. Então, é possível deduzir e selecionar, por exemplo, de acordo com os axiomas da ontologia, quais ferramentas podem gerar carga nos protocolos da aplicação, quais atividades devem ser executadas para atingir o objetivo do teste e quais métricas podem ser monitoradas pelas ferramentas escolhidas. De acordo com [24] existem aproximadamente 300 ferramentas de teste atualmente, o que dificulta a identificação da ferramenta mais adequada para um determinado teste. Portanto, se um testador possuir conhecimento sobre as tecnologias utilizadas em testes de software, este testador pode recomendar as melhores opções para o ambiente a ser testado. Pensando nisto, um dos objetivos da ontologia proposta é modelar este conhecimento para que até mesmo pessoas sem experiência em teste de software possam selecionar opções coerentes com seus objetivos específicos de teste.

UC3 – Gerenciar instâncias dos conceitos do domínio. O caso de uso “Gerenciar instâncias dos conceitos do domínio” deve permitir a criação, recuperação e atualização de instâncias dos conceitos e propriedades representados na ontologia. Como comentado anteriormente, é possível imaginar que a ontologia possuirá conceitos para representar os testes de desempenho, as ferramentas, as atividades, o ambiente de teste, as métricas de desempenho, entre outros. Portanto, uma aplicação baseada nesta ontologia proposta deve permitir criar, pesquisar e editar instâncias destes conceitos.

UC4 – Criar plano de teste de desempenho. O caso de uso “Criar plano de teste de desempenho” se refere à criação de um documento contendo os axiomas relacionados a uma instância do conceito que representa o teste de desempenho. Portanto, para criar um plano de teste, é preciso possuir uma instância do teste, como indica a relação *<<include>>* no diagrama de casos de uso da Figura 15. Segundo [27], o plano de teste pode ser utilizado para aumentar o gerenciamento do teste e a visibilidade do processo. O plano de teste define o escopo, a abordagem, os recursos, as características e itens a serem testados, as atividades de teste a serem executadas, o cronograma com os responsáveis pelas atividades e os riscos associados [27]. Segundo [24], o plano de teste pode incluir a aplicação ou sistema sob teste, os objetivos do teste (incluindo requisitos e riscos), o escopo e as limitações do teste, o ambiente de teste, a estratégia de teste, os detalhes para cada fase de desenvolvimento e o cronograma. De acordo com [24], um esboço inicial do plano de teste deve ser criado ao final da atividade de análise da fase de desenvolvimento e atualizado iterativamente durante as fases de desenvolvimento subsequentes. Portanto, o plano de teste é desenvolvido de forma iterativa, sendo atualizado conforme novas informações sobre a aplicação e os objetivos de teste se tornam disponíveis.

Tendo uma ideia sobre o escopo da ontologia e o que uma aplicação baseada nela deve fazer, a etapa seguinte consiste em conceitualizar o domínio, ou seja, definir as classes, propriedades, axiomas e instâncias da ontologia.

4.2. Conceitos, Propriedades e Instâncias da Ontologia

Após definir o escopo da ontologia, a metodologia utilizada [49] sugere o reuso de ontologias existentes. Assim, ontologias de domínios relacionados, como a OntoTest [19] (*Ontology of Software Testing*) e a SwTO [13] (*Software Test Ontology*) foram estudadas para o desenvolvimento deste trabalho. A ontologia proposta foi construída também com base em conceitos extraídos, sobretudo, do Corpo de Conhecimento em Engenharia de Software (SWEBOK) [1], do glossário da Engenharia de Software (IEEE Std. 610.12) [28] e do padrão para documentação do teste de software (IEEE Std. 829) [27]. Estas referências foram escolhidas para minimizar o viés, proporcionar uma base teórica mais sólida e garantir a coerência entre os conceitos e as propriedades da ontologia. Além disso, os trabalhos que utilizam ontologias no teste de software, analisados na Seção 3 desta dissertação, também foram considerados no desenvolvimento e aplicação da

ontologia proposta. A ontologia foi representada na língua inglesa para aumentar sua abrangência e seguir o padrão adotado na OntoTest [19] e na SwTO [13].

De acordo com a metodologia adotada [49], os principais termos da ontologia podem ser definidos como os termos sobre os quais se deseja fazer declarações ou explicar para um usuário, que no caso é um testador de desempenho. Em outras palavras, a metodologia sugere pensar em “Quais termos gostaríamos de falar sobre?” e “O que gostaríamos de dizer sobre estes termos?”. Como a ontologia deve especificar um teste de desempenho, espera-se que a ontologia defina, por exemplo, por que fazer um teste, como fazê-lo, o que deve ser feito e quais resultados podem ser obtidos. Assim, o teste de desempenho, as atividades dos testes, as ferramentas de teste, os objetivos dos testes e as métricas de desempenho compõem os principais termos, estando de acordo com as questões de competência e os casos de uso que representam o escopo da ontologia. Portanto, os conceitos principais da ontologia foram definidos, como ilustrado na Figura 16 e detalhados a seguir:

- ***PTActivity*** – O conceito *PTActivity* representa uma atividade de teste, que é parte do processo de execução de um teste de desempenho.
- ***PTArtifact*** – Este conceito representa um artefato ou entregável de teste, ou seja, um produto de trabalho gerado a partir da execução de uma atividade de teste (*PTActivity*).
- ***SystemUnderTestConcept*** – O conceito *SystemUnderTestConcept* possui subclasses que especificam o alvo de um teste de desempenho, ou seja, o sistema sob teste.
- ***PerformanceTest*** – Como esperado de uma ontologia sobre o domínio de teste de desempenho, existe um conceito responsável por representar os testes de desempenho e este conceito foi denominado *PerformanceTest*.
- ***PerformanceTester*** – O conceito *PerformanceTester* representa um testador, que é o responsável pela execução das atividades do teste de desempenho (*PTActivity*).
- ***PerformanceMetric*** – O conceito *PerformanceMetric* é utilizado para representar as métricas de desempenho, que são monitoradas pelas ferramentas de teste (*PTTool*).
- ***ValuePartition*** – O conceito *ValuePartition* representa conceitos que não pertencem ao domínio da ontologia, mas são utilizados por conceitos pertencentes ao domínio. Esta abordagem faz parte de um padrão de desenvolvimento de ontologias adotado, que separa os conceitos em *DomainConcept* ou *ValuePartition*.
- ***PTGoal*** – O conceito *PTGoal* representa uma razão ou objetivo que justifica a execução de um teste de desempenho (*PerformanceTest*).
- ***PTTool*** – Uma ferramenta de teste (*PTTool*) oferece recursos para apoiar a execução de atividades (*PTActivity*), porém cada ferramenta possui restrições de funcionamento.

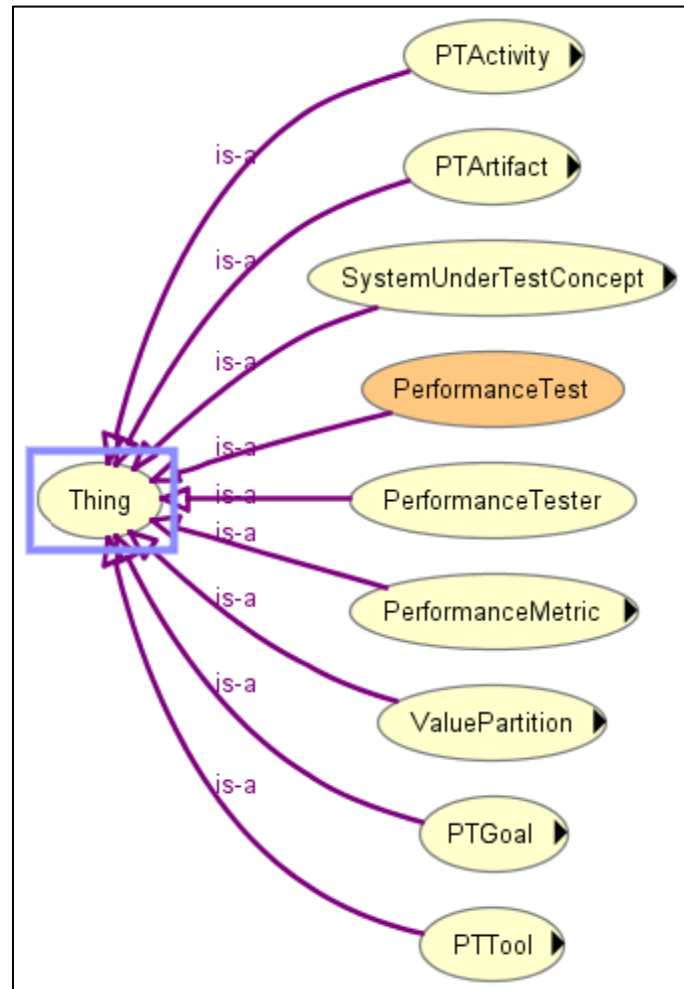


Figura 16: Hierarquia dos conceitos do domínio (imagem gerada com o plugin OWLViz do Protégé).

É importante observar na Figura 16 que o conceito *PerformanceTest* recebeu uma cor diferente em comparação aos outros conceitos. Isto se deve ao fato do conceito teste de desempenho ter sido representado como uma classe definida, enquanto que os outros conceitos da ontologia são classes primitivas. A linguagem OWL permite que os conceitos da ontologia sejam representados como classes definidas ou primitivas [59]. Classes primitivas possuem restrições que todos os indivíduos pertencentes à classe devem satisfazer, mas não significa que um indivíduo aleatório que satisfaça estas condições necessariamente pertencerá àquela classe. Já uma classe definida significa tanto que os indivíduos da classe satisfazem suas restrições como que qualquer indivíduo aleatório que satisfaça tais restrições pode ser classificado como pertencente àquela classe [59]. Estas características definem a semântica da linguagem OWL e são levadas em consideração, por exemplo, quando um *reasoner* é aplicado sobre a ontologia. Um *reasoner* é um software capaz de inferir consequências lógicas com base em um conjunto de axiomas. Os axiomas do conceito *PerformanceTest* estão ilustrado na Figura 17 como “*Necessary & Sufficient Asserted Conditions*”. Por exemplo, um destes axiomas define

que uma instância de *PerformanceTest* deve possuir pelo menos um relacionamento com um indivíduo do tipo *PTActivity* por meio da propriedade *hasActivity*. Ainda, um teste de desempenho deve possuir relacionamentos com pelo menos uma instância dos conceitos *PTGoal*, *ApplicationUnderTest*, *PerformanceMetric* e *PTTool* por meio das propriedades correspondentes. Alguns conceitos da ontologia utilizam o acrônimo *PT* como abreviação de *Performance Test* (Teste de Desempenho).

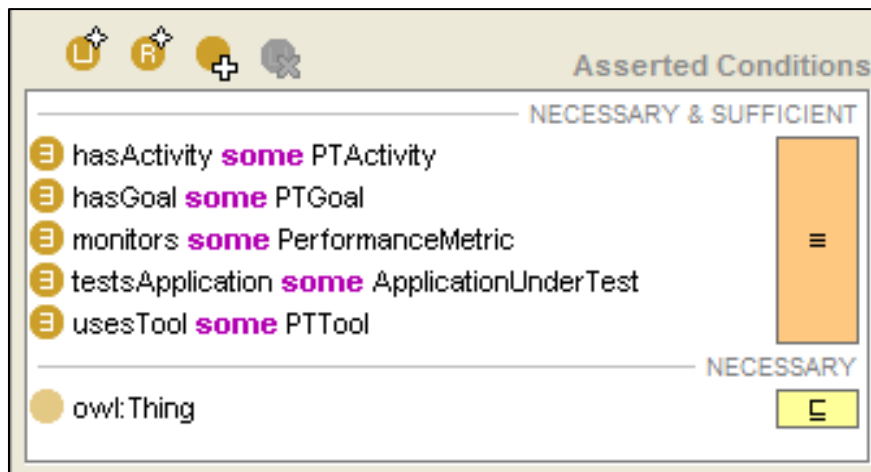


Figura 17: Restrições sobre o conceito *PerformanceTest*.

Classes definidas, como é o caso de *PerformanceTest*, apresentam suas restrições como “*Necessary & Sufficient*” (Figura 17). Em contrapartida, as restrições das classes primitivas são representadas no item “*Necessary*”, como é o caso da Figura 18, que apresenta os objetivos dos teste de desempenho e as restrições sobre o conceito *NormalWorkloadPerformanceEvaluation*. Se o objetivo do teste é avaliar o desempenho do sistema com uma carga normal (semelhante a carga real que o sistema receberá dos usuários), então deve ser executada pelo menos uma atividade de análise das medições de desempenho (*PerformanceMeasurementAnalysis*). Esta restrição foi representada pela propriedade *imposesActivity*, cujo domínio é o conceito *PTGoal* e a imagem é *PTActivity*. Cada objetivo pode possuir diferentes restrições e, como ilustrado na Figura 18, a ontologia possui 7 conceitos que representam os objetivos dos testes de desempenho:

- ***DifferentConfigurationsComparison*** – Este conceito representa o objetivo de executar o mesmo caso de teste em diferentes configurações para comparar determinada métrica de desempenho (também pode ser chamado de *benchmarking*).
- ***LongRunPerformanceEvaluation*** – Este objetivo simboliza a execução de um caso de teste durante intervalos maiores de tempo para avaliar se existe a degradação das métricas de desempenho ao longo do tempo (também chamado de *endurance test*).

- **NormalWorkloadPerformanceEvaluation** – Representa a execução de um teste utilizando uma carga normal, *i.e.*, semelhante a carga que o sistema será submetido por usuários reais quando estiver em produção (também chamado de *load test*).
- **PTArtifactElaboration** – Representa um teste de desempenho que possui como objetivo a elaboração de um entregável, que pode ser um caso de teste, um relatório, uma especificação, a medição de uma métrica de desempenho, entre outros.
- **PerformanceBottleneckIdentification** – Este objetivo simboliza a execução de um teste visando encontrar os gargalos do sistema, sendo necessário utilizar cargas altas para identificar qual recurso atinge seu limite primeiro (também chamado de *stress test*).
- **PerformanceImprovement** – Este conceito define o objetivo de executar um teste de desempenho com a finalidade de melhorar o desempenho do sistema, o que geralmente se torna possível após a identificação dos gargalos do sistema.
- **PerformanceRequirementsVerification** – A verificação dos requisitos de desempenho é também um objetivo que os testes podem possuir, sendo estes requisitos usualmente definidos com base nas métricas de desempenho.

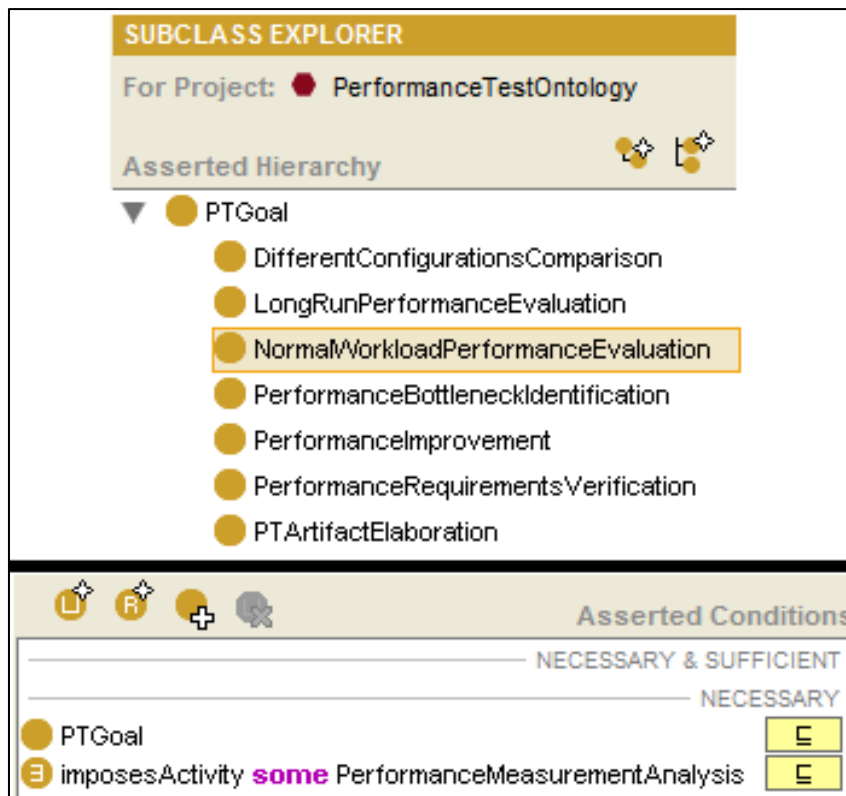


Figura 18: Conceitos que especializam *PTGoal* e restrições sobre *NormalWorkloadPerformanceEvaluation*.

Os objetivos foram modelados como conceitos na ontologia para permitir a especificação de restrições em OWL, como é o caso, por exemplo, da restrição explicada

anteriormente que utiliza a propriedade *imposesActivity*. Porém, cada um destes conceitos só precisa possuir uma única instância, dado que esta única instância denomina o objetivo do teste e pode ser referenciada por diversas instâncias de *PerformanceTest*.

A Figura 19 apresenta a hierarquia das atividades dos testes de desempenho, que foi subdividida em 4 categorias: atividades de desenvolvimento (*PTDevelopmentActivity*), de documentação (*PTDocumentationActivity*), de execução (*PTExecutionActivity*) ou de planejamento (*PTPlanningActivity*).

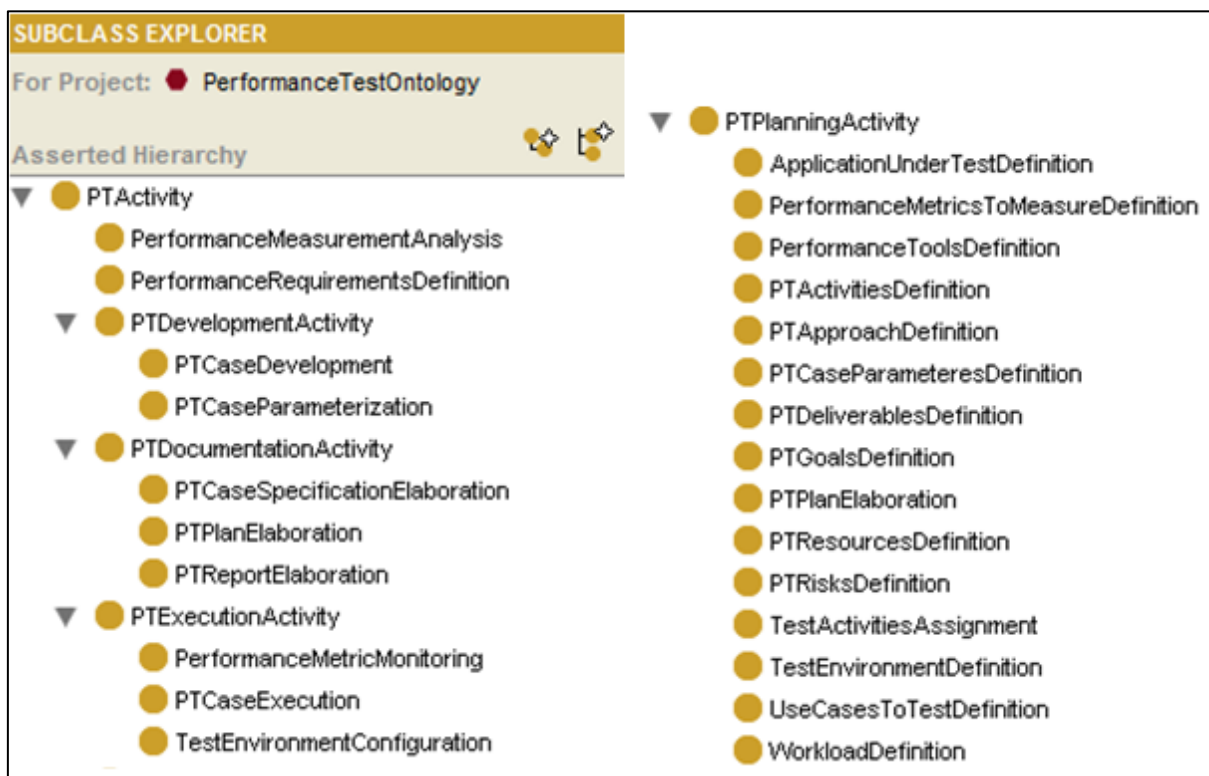


Figura 19: Hierarquia das atividades dos testes de desempenho adotada na ontologia.

As atividades de desenvolvimento (*PTDevelopmentActivity*) focam na geração ou modificação de um caso de teste. São exemplos os conceitos *PTCaseDevelopment* e *PTCaseParameterization*, responsáveis por representar, respectivamente, as atividades de criação e parametrização de um caso de teste de desempenho. A Figura 20 apresenta as restrições do conceito *PTCaseDevelopment*, que permitem afirmar que qualquer instância do tipo *PTCaseDevelopment* necessariamente:

- será também do tipo *PTDevelopmentActivity*;
- depende da execução das atividades *WorkloadDefinition*, *TestEnvironmentDefinition*, *UseCasesToTestDefinition* e *PerformanceToolsDefinition*; e
- gera como resultado de sua execução uma instância do conceito *PTCase*.

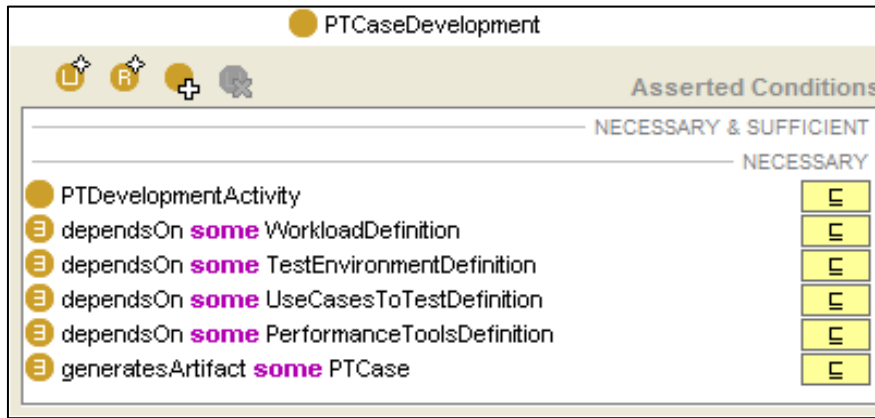


Figura 20: Restrições do conceito *PTCaseDevelopment*.

É importante observar que cada conceito que especializa *PTActivity* pode ser definido com base em diferentes restrições. Estas restrições de classes utilizam as propriedades da ontologia que possuem como domínio o conceito *PTActivity*. Um exemplo é a propriedade transitiva *dependsOn*, que possui *PTActivity* como domínio e como imagem. Devido à característica transitividade, se a atividade A depende da atividade B e a atividade B depende da atividade C, então se pode deduzir que A depende de C. O conceito *PTActivity* é domínio também da propriedade *generatesArtifact*, responsável por representar que uma atividade pode gerar uma instância do tipo entregável (*PTArtifact*).

O conceito atividade de documentação (*PTDocumentationActivity*) representa as atividades que possuem como característica a finalidade de documentar os artefatos relacionados ao teste. A Figura 21 apresenta as restrições de dois conceitos que especializam *PTDocumentationActivity*, permitindo comparar suas diferentes restrições.

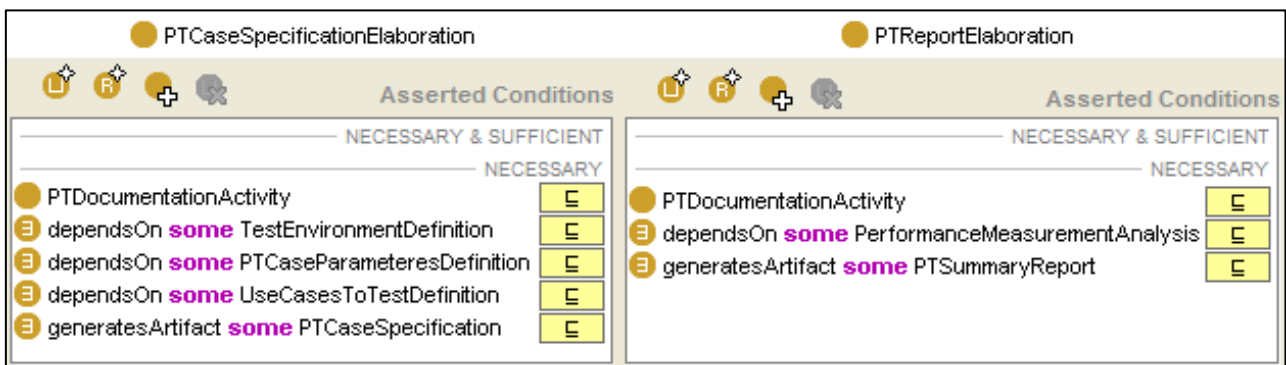


Figura 21: Restrições dos conceitos *PTCaseSpecificationElaboration* e *PTReportElaboration*.

As atividades de execução possuem como característica o foco na execução de um caso de teste. São exemplos de *PTExecutionActivity*, como ilustrado na Figura 22, as atividades *PTCaseExecution* e *PerformanceMetricMonitoring*.

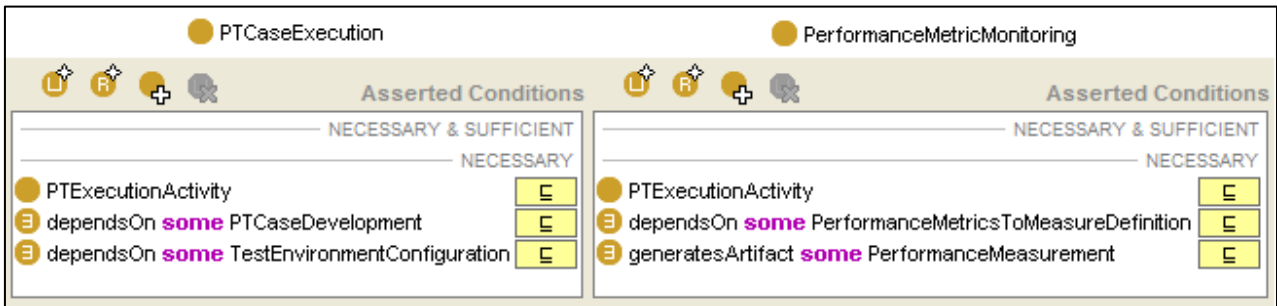


Figura 22: Restrições dos conceitos *PTCaseExecution* e *PerformanceMetricMonitoring*.

As atividades de planejamento são responsáveis pela identificação e definição das características a nível de planejamento do teste de desempenho. A Figura 23 apresenta as restrições de dois conceitos do tipo *PTPlanningActivity*: *PerformanceToolsDefinition* e *WorkloadDefinition*.

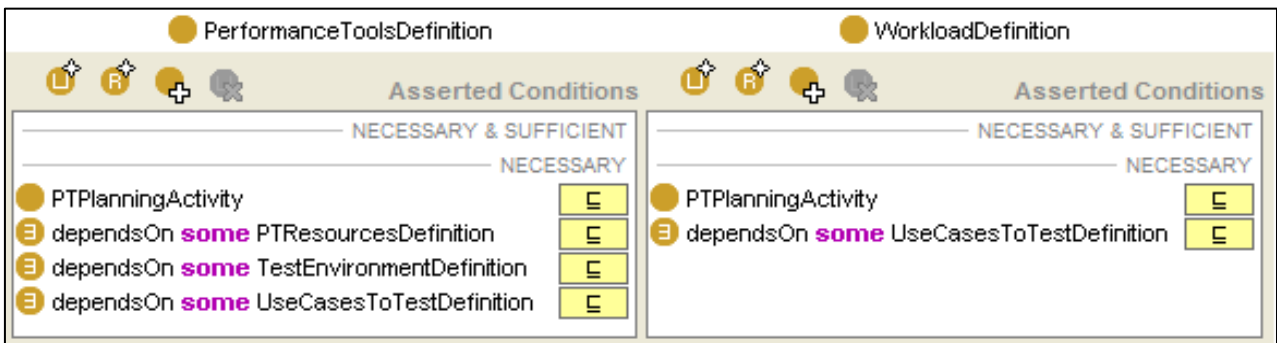


Figura 23: Restrições dos conceitos *PerformanceToolsDefinition* e *WorkloadDefinition*.

Estas 4 categorias de atividades de teste não são mutuamente excludentes, como é o exemplo da atividade de elaboração do plano de teste (*PTPlanElaboration*) que foi classificada simultaneamente como atividade de documentação e de planejamento. É importante observar também que existem 2 subconceitos de *PTActivity* (denominados *PerformanceRequirementsDefinition* e *PerformanceMeasurementAnalysis*) que não estão classificados em nenhuma destas 4 categorias apresentadas anteriormente.

Um outro conceito principal da ontologia proposta foi denominado *PTArtifact* e é responsável por representar os artefatos, entregáveis ou produtos de trabalho dos testes de desempenho. Como ilustrado na Figura 24, dentre os artefatos gerados no teste de software encontra-se as medições de desempenho, os casos de testes e a documentação. A ontologia proposta pode auxiliar o testador a definir qual conjunto de documentos de teste serão usados e o que cada um destes documentos deve conter. Os conceitos da ontologia fornecem uma descrição padrão dos documentos de teste que

pode servir de referência e facilitar a comunicação sobre o significado dos artefatos de teste, como, por exemplo, o plano de teste. A taxonomia dos artefatos de teste desempenho na ontologia seguiu o padrão 829 definido pela IEEE [27], que divide a documentação de teste em planejamento, especificação ou relato dos resultados. A explicação para os conceitos da hierarquia de *PTArtifact* é a seguinte:

- ***PTPlan***. O plano de teste determina o escopo, a abordagem, os recursos, os itens a serem testados, as atividades de testes a serem realizados, os responsáveis por cada tarefa e os riscos associados [27].

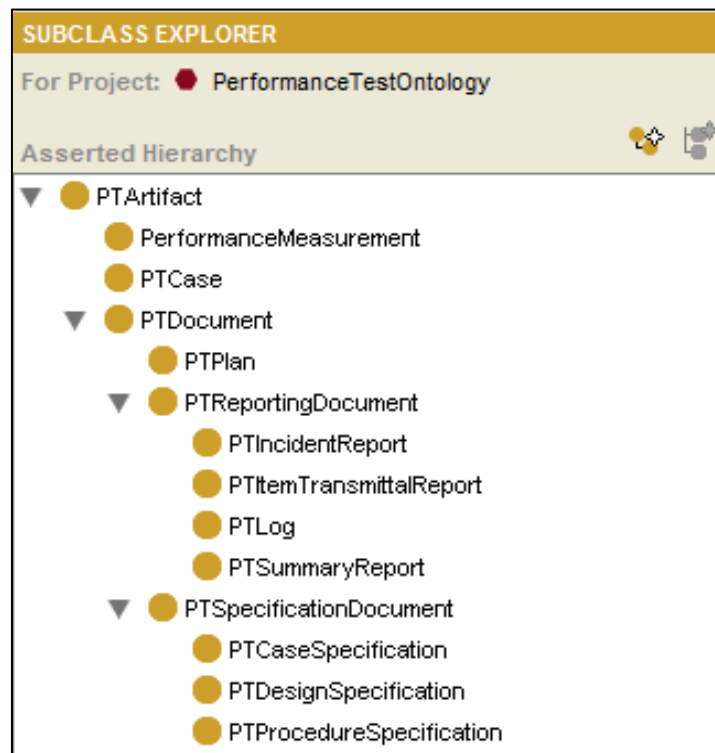


Figura 24: Hierarquia de conceitos dos entregáveis dos testes de desempenho.

- ***PTIncidentReport***. O relatório de incidentes é utilizado para descrever eventos que ocorreram durante a execução do teste e que necessitam de investigação adicional [27].

- ***PTItemTransmittalReport***. Este relatório identifica itens de teste sendo transmitidos para o teste no evento que marca o início da fase de execução dos testes, separando do desenvolvimento do teste [27].

- ***PTLog***. Um log do teste é utilizado para registrar acontecimentos que ocorreram durante a execução do teste [27].

- ***PTSummaryReport***. Este relatório contém uma avaliação dos itens de teste correspondentes e um resumo das atividades de teste associadas a uma ou mais especificações de projeto de teste [27].

- ***PTCaseSpecification***. A especificação de um caso de teste documenta os valores de entrada, os valores esperados de saída e as restrições sobre a execução de um item de teste [27]. Segundo [24], a documentação de um caso de teste pode incluir: a fase do desenvolvimento em que o caso de teste será executado, os objetivos específicos de teste, os dados de teste sugeridos, as ferramentas de teste, os procedimentos de inicialização, finalização e reinicialização do teste, as etapas de execução do teste (cada etapa possui uma descrição das ações, resultados esperados e resultados obtidos) e a lista de defeitos do software encontrados neste caso de teste.
- ***PTDesignSpecification***. A especificação do projeto de teste é responsável por refinar a abordagem de teste, especificar os critérios de aprovação/falha, identificar as características a serem cobertas pelo projeto e destacar os casos e os procedimentos necessários para a realização do teste [27].
- ***PTProcedureSpecification***. A especificação do procedimento de teste identifica os passos necessários para operar o sistema e exercitar os casos de teste definidos, os quais implementam o projeto de teste associado [27].

As métricas de desempenho também estão entre os principais conceitos modelados na ontologia, agrupadas conceitualmente em duas categorias: *MachinePerformanceMetric* (métricas referentes aos recursos das máquinas) e *PTMetric* (relacionadas a execução do teste). As métricas de máquina são divididas em conceitos mais especializados, como *DiskMetric*, *MemoryMetric*, *NetworkMetric*, *ProcessMetric* e *ProcessorMetric*, utilizados para representar, respectivamente, as métricas de disco, memória, rede, processo e processador. Como ilustrado na Figura 25, estes conceitos já possuem instâncias, como por exemplo, a quantidade de memória disponível e o número de páginas acessadas por segundo são instâncias do conceito *MemoryMetric*, que possui ao total 11 instâncias. Ainda em relação às métricas, a ontologia possui a propriedade *canMonitor* para representar que uma instância de ferramenta de teste pode monitorar determinados indicadores de desempenho, ou seja, a propriedade *canMonitor* possui o conceito *PTTool* como domínio e *PerformanceMetric* como imagem. A conceitualização das métricas de desempenho foi baseada em [9], onde foi definida uma taxonomia para especificar parâmetros relacionados à Qualidade de Serviço. Em [9], as métricas referentes a Qualidade de Serviço foram classificadas de acordo com três perspectivas: aplicação, recurso ou sistema. O tempo de resposta de uma transação ou o número de erros são exemplos de métricas do ponto de vista da aplicação e a utilização de processadores, memória e discos são métricas a nível de recursos [9].

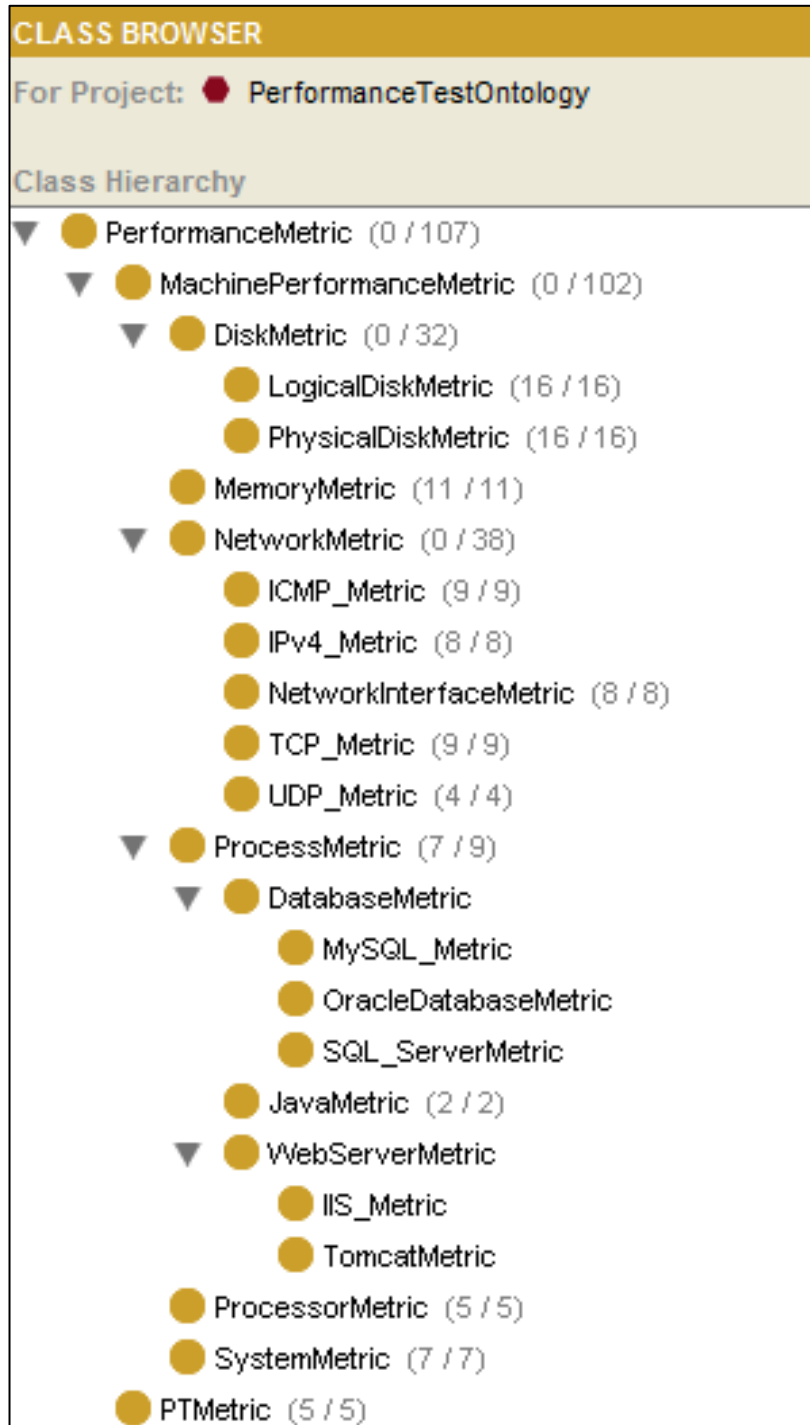


Figura 25: Hierarquia e instâncias das subclasses de *PerformanceMetric*.

As instâncias dos conceitos que especializam *PerformanceMetric*, apresentadas na Tabela 1, foram extraídas segundo as capacidades de monitoramento das ferramentas de teste. É importante citar que é possível expandir o conhecimento da ontologia com novas instâncias e conceitos que poderão ser adicionados futuramente.

Tabela 1: Principais instâncias de conceitos que especializam o conceito *PerformanceMetric*.

Conceito	Instâncias
PTMetric	RequestsPerSecond, ResponseSizeBytes, ResponseTimeSeconds, ThroughputMegabytes, TransactionsPerSecond
SystemMetric	ContextSwitchesPerSecond, FileReadBytesPerSecond, FileWriteBytesPerSecond, ProcessorQueueLenght, FileWriteOperationsPerSecond, SystemCallsPerSecond, FileReadOperationsPerSecond
ProcessorMetric	InterruptsPerSecond, PercentageOfInterruptTime, PercentageOfPrivilegedTime, PercentageOfUserTime, PercentageOfProcessorTime
MemoryMetric	AvailableMegabytes, PageWritesPerSecond, CommittedBytes, CacheFaultsPerSecond, FreeSystemPageTableEntries, PageFaultsPerSecond, PagesPerSecond, PoolPagedBytes, PoolNonPagedBytes, PageReadsPerSecond, CacheBytes
ProcessMetric	ProcessPageFaultsPerSecond, ProcessPrivateBytes, ProcessPercentageOfProcessorTime, ProcessThreadCount, ProcessPoolNonPagedBytes, ProcessPoolPagedBytes, ProcessPercentageOfUserTime
PhysicalDiskMetric	PDAverageBytesPerRead, PDAverageBytesPerWrite, PDAverageQueueLenght, PDAverageReadQueueLenght, PDAverageSecondsPerRead, PDAverageSecondsPerWrite, PDAverageWriteQueueLenght, PDBytesReadPerSecond, PDBytesWritePerSecond, PDCurrentQueueLenght, PDPercentageOfDiskReadTime, PDPercentageOfDiskTime, PDPercentageOfDiskWriteTime, PDPercentageOfFreeSpace, PDReadsPerSecond, PDWritesPerSecond
NetworkInterfaceMetric	BytesReceivedPerSecond, BytesSentPerSecond, CurrentBandwidth, BytesTotalPerSecond, OutputQueueLenght, PacketsSentPerSecond, PacketsReceivedPerSecond, PacketsTotalPerSecond
ICMP_Metric	EchoReplyReceivedPerSecond, EchoReplySentPerSecond, EchoRequestReceivedPerSecond, MessagesPerSecond, MessagesSentPerSecond, RedirectSentPerSecond, RedirectReceivedPerSecond, EchoRequestSentPerSecond
IPv4_Metric	FragmentedDatagramsPerSecond, FragmentsCreatedPerSecond, FragmentsReassembledPerSecond, FragmentsReceivedPerSecond, IPDatagramsForwardedPerSecond, IPDatagramsPerSecond, IPDatagramsReceivedPerSecond, IPDatagramsSentPerSecond
TCP_Metric	ConnectionFailure, ConnectionsActive, ConnectionsEstablished, ConnectionsPassive, ConnectionsReset, SegmentsPerSecond, SegmentsReceivedPerSecond, SegmentsRetransmittedPerSecond, SegmentsSentPerSecond

As ferramentas de teste (*PTTool*) estão também entre os principais conceitos do domínio de teste de desempenho. A Figura 26 ilustra as 18 instâncias e as 9 subclasses do conceito *PTTool*. As instâncias podem ser visualizadas pelos losangos de cor roxa na direita da Figura 26 e as subclasses estão detalhadas na Figura 27. Além disso, o

conceito *PTTool* é domínio das propriedades *canCollectMetric* com imagem em *PerformanceMetric*, *canMonitor* e *worksOn* com imagem no conceito *OperatingSystem*, *generatesLoadOn* com imagem em *CommunicationProtocol* e *supports* cuja imagem é o conceito *PTActivity*.

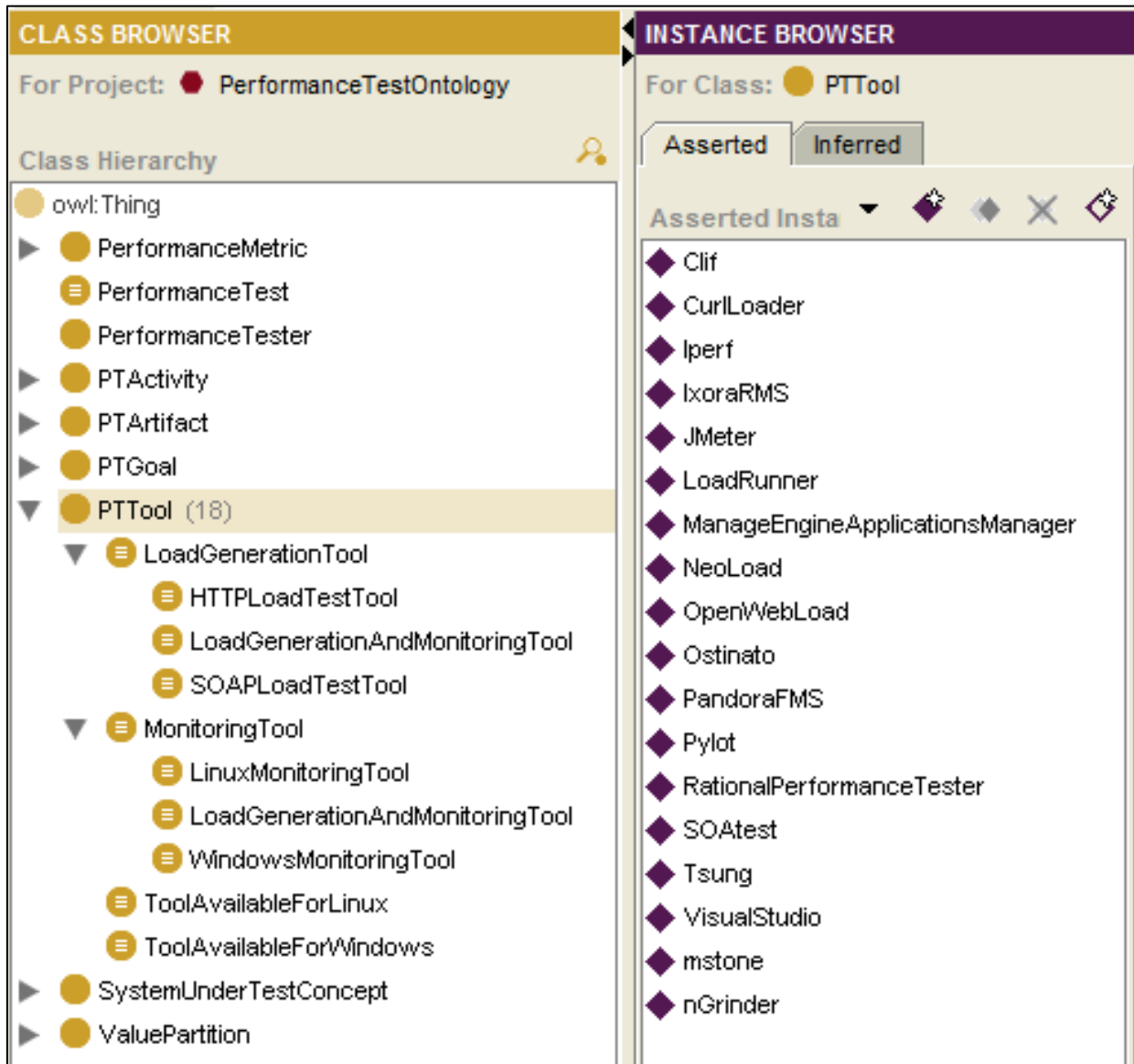


Figura 26: Hierarquia de conceitos e instâncias das ferramentas de teste de desempenho.

As restrições dos conceitos que especializam *PTTool* estão ilustradas na Figura 27. Por exemplo, se uma ferramenta de teste (instância de *PTTool*) possuir pelo menos um relacionamento com uma instância de *CommunicationProtocol* por meio da propriedade *generateLoadOn*, então é possível classificar esta instância como uma ferramenta capaz de gerar carga (*LoadGenerationTool*). Da mesma forma, o conceito *HTTPLoadTestTool* representa as ferramentas que podem gerar carga no protocolo HTTP; o conceito *MonitoringTool* agrupa ferramentas que podem monitorar pelo menos uma instância de

MachinePerformanceMetric; o conceito *LinuxMonitoringTool* classifica as ferramentas que podem monitorar pelo menos uma instância do sistema operacional Linux; entre outros conceitos ilustrados na Figura 27.

Conceito	Restrições do conceito no Protégé
<ul style="list-style-type: none"> LoadGenerationTool 	<ul style="list-style-type: none"> PTTool generatesLoadOn some CommunicationProtocol
<ul style="list-style-type: none"> HTTPLoadTestTool 	<ul style="list-style-type: none"> LoadGenerationTool generatesLoadOn has HTTP
<ul style="list-style-type: none"> SOAPLoadTestTool 	<ul style="list-style-type: none"> LoadGenerationTool generatesLoadOn has SOAP
<ul style="list-style-type: none"> MonitoringTool 	<ul style="list-style-type: none"> PTTool canCollectMetric some MachinePerformanceMetric supports some PerformanceMetricMonitoring
<ul style="list-style-type: none"> LinuxMonitoringTool 	<ul style="list-style-type: none"> MonitoringTool canMonitor some LinuxBasedOperatingSystem
<ul style="list-style-type: none"> WindowsMonitoringTool 	<ul style="list-style-type: none"> MonitoringTool canMonitor some WindowsBasedOperatingSystem
<ul style="list-style-type: none"> LoadGenerationAndMonitoringTool 	<ul style="list-style-type: none"> LoadGenerationTool MonitoringTool
<ul style="list-style-type: none"> ToolAvailableForLinux 	<ul style="list-style-type: none"> PTTool worksOn some LinuxBasedOperatingSystem
<ul style="list-style-type: none"> ToolAvailableForWindows 	<ul style="list-style-type: none"> PTTool worksOn some WindowsBasedOperatingSystem

Figura 27: Restrições sobre os conceitos utilizados pelo *reasoner* para classificar instâncias de *PTTool*.

Os conceitos definidos apresentados na Figura 27 não possuem instâncias, contudo um *reasoner* pode ser aplicado para inferir se alguma instância da ontologia pode ser classificada nestes conceitos. Foi aplicado o *reasoner* Pellet [22] sobre os conceitos e instâncias da ontologia e o resultado da inferência pode ser visualizado na seção 5.4. Por exemplo, das 18 instâncias de *PTTool*, 15 foram classificadas como *LoadGenerationTool*, 14 instâncias são do tipo *HTTPLoadTestTool*, 10 foram classificadas como *MonitoringTool* e assim por diante. É esperado de um *reasoner* OWL as funcionalidades de verificação de consistências, satisfatibilidade dos conceitos, classificação e realização [21].

Como ilustrado na Figura 28, o conceito *SystemUnderTestConcept*, utilizado para especificar as características do sistema sob teste, possui as seguintes subclasses:

ApplicationUnderTest, *ApplicationUseCase*, *CommunicationProtocol* e *OperatingSystem*. O conceito *ApplicationUnderTest* representa a aplicação sob teste e *ApplicationUseCase* simboliza os casos de uso da aplicação a serem testados. É importante observar que estes dois conceitos ainda não possuem instâncias, uma vez que devem ser instanciados durante a elaboração de um teste de desempenho. Já o conceito *CommunicationProtocol* possui 20 instâncias que se relacionam com instâncias dos conceitos *PTTool* e *ApplicationUseCase*. Em outras palavras, os protocolos são utilizados nas aplicações e as ferramentas de teste conseguem gerar carga em determinados protocolos.

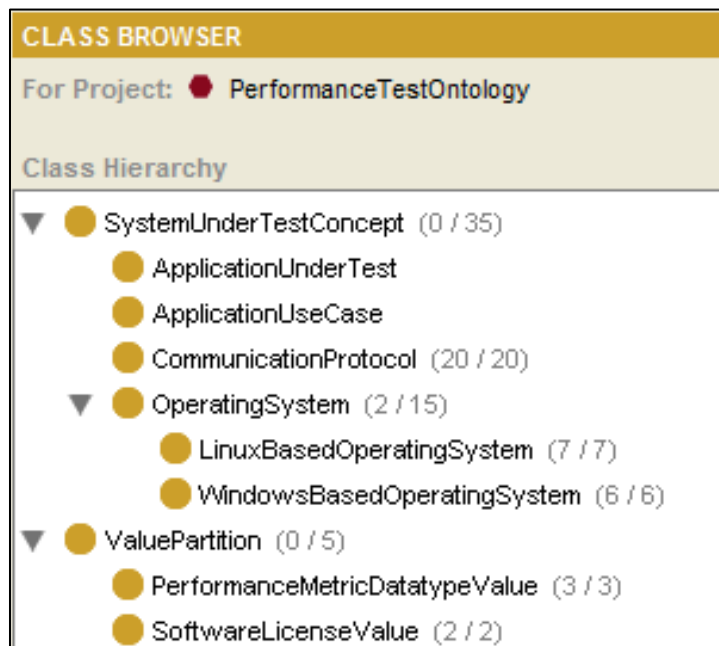


Figura 28: Hierarquia dos conceitos do sistema sob teste e padrão *ValuePartition*.

O conceito *CommunicationProtocol* apresenta instâncias como, por exemplo, ARP, DNS, FTP, HTTP, ICMPv4, ICMPv6, IMAP, IPv4, JMS, NFS, RPC, SMPT, SNMP, SOAP, SSH, TCP, UDP. O conceito *OperatingSystem* possui duas instâncias (Mac e Solaris) e duas subclasses: uma denominada *LinuxBasedOperatingSystem* (possui as instâncias Debian, Fedora, Gentoo, Mandriva, OpenSuse, RedHat, Ubuntu); e outra subclasse denominada *WindowsBasedOperatingSystem* (apresenta as instâncias WindowsXP, WindowsVista, Windows7, Windows8, WindowsServer2003, WindowsServer2008). Ainda, foi utilizado um padrão de desenvolvimento de ontologias denominado Value Partition¹. De acordo com este padrão, o conceito *ValuePartition* possui subclasses que são utilizadas para especificar características dos conceitos que pertencem ao domínio da ontologia. O conceito *PerformanceMetricDatatypeValue* é uma destas subclasses, sendo

¹ Mais informações sobre este padrão podem ser encontradas em <http://www.w3.org/TR/swbp-specified-values/>

utilizado para especificar o tipo de dado de uma instância de *PerformanceMetric* e possuindo as instâncias *IntegerNumber*, *FloatingNumber* e *String*. O conceito *SoftwareLicenseValue* é outro exemplo de conceito do tipo *ValuePartition*, que é utilizado para especificar a licença de uso e comercialização de uma instância de *PTTool*, permitindo os valores *OpenSourceLicense* e *ProprietaryLicense*.

Embora muitas propriedades da ontologia já tenham sido mencionadas ao longo das explicações anteriores acerca dos conceitos, uma lista completa das propriedades existentes pode ser visualizada na Figura 29. Além disso, uma explicação detalhada destas propriedades está disponível a seguir:

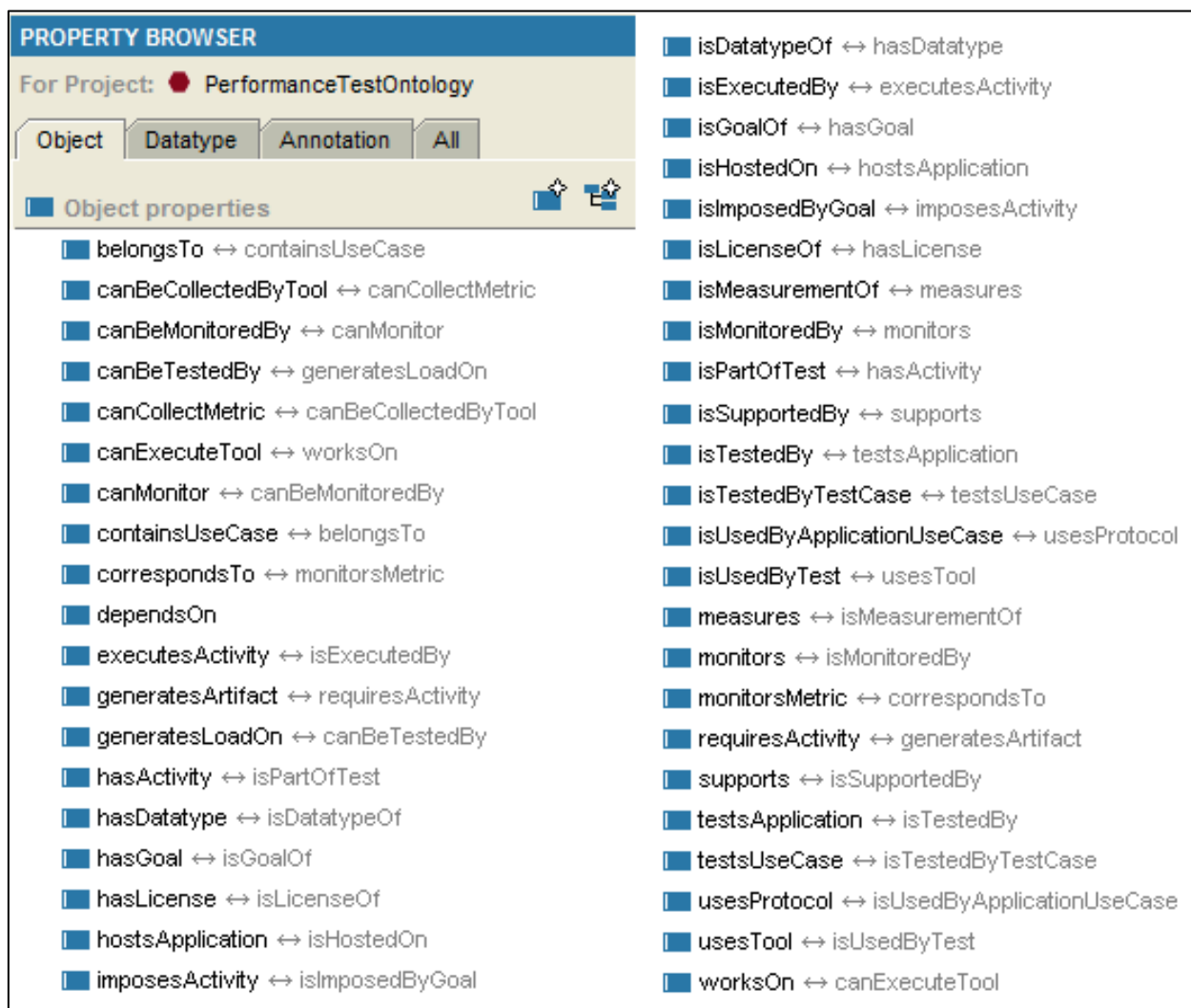


Figura 29: Propriedades representadas na ontologia.

Propriedade: *containsUseCase* (inversa de *belongsTo*)

Domínio: *ApplicationUnderTest*

Imagem: *ApplicationUseCase*

Significado: Esta propriedade representa a informação de que uma aplicação sobre teste pode possuir múltiplos casos de uso a serem testados.

Propriedade: *canCollectMetric* (inversa de *canBeCollectedByTool*)

Domínio: *PTTool*

Imagem: *PerformanceMetric*

Significado: Contém a informação de que uma instância de ferramenta de teste pode coletar métricas de múltiplas instâncias do conceito *PerformanceMetric*.

Propriedade: *canMonitor* (inversa de *canBeMonitoredBy*)

Domínio: *PTTool*

Imagem: *OperatingSystem*

Significado: A propriedade *canMonitor* armazena a informação de quais sistemas operacionais é possível monitorar com cada uma das ferramentas de teste.

Propriedade: *generatesLoadOn* (inversa de *canBeTestedBy*)

Domínio: *PTTool*

Imagem: *CommunicationProtocol*

Significado: Representa o conhecimento de quais protocolos de comunicação é possível gerar carga utilizando cada uma das ferramentas de teste de desempenho.

Propriedade: *canExecuteTool* (inversa de *worksOn*)

Domínio: *OperatingSystem*

Imagem: *PTTool*

Significado: Esta propriedade é utilizada para selecionar em quais sistemas operacionais é possível executar uma ferramenta de teste específica.

Propriedade: *monitorsMetric* (inversa de *correspondsTo*)

Domínio: *ApplicationUseCase*

Imagem: *PTMetric*

Significado: Representa quais métricas a nível de teste devem ser monitoradas para cada caso de uso da aplicação a ser testada.

Propriedade: *dependsOn*

Domínio: *PTActivity*

Imagem: *PTActivity*

Significado: Simboliza a relação transitiva de dependência entre as atividades executadas nos testes de desempenho.

Propriedade: *executesActivity* (inversa de *isExecutedBy*)

Domínio: *PerformanceTester*

Imagem: *PTActivity*

Significado: Contém a informação dos responsáveis pela execução de cada atividade.

Propriedade: *generatesArtifact* (inversa de *requiresActivity*)

Domínio: *PTActivity*

Imagem: *PTArtifact*

Significado: A propriedade *generatesArtifact* é utilizada para especificar quais entregáveis cada atividade de teste pode gerar.

Propriedade: *hasActivity* (inversa de *isPartOfTest*)

Domínio: *PerformanceTest*

Imagem: *PTActivity*

Significado: Contém a informação das atividades de teste a serem executadas em uma instância específica de teste de desempenho.

Propriedade: *hasDatatype* (inversa de *isDatatypeOf*)

Domínio: *PerformanceMetric*

Imagem: *PerformanceMetricDatatypeValue*

Significado: Esta propriedade mapeia instâncias de métricas de desempenho com um tipo de dado representado por instâncias do conceito *PerformanceMetricDatatypeValue*.

Propriedade: *hasGoal* (inversa de *isGoalOf*)

Domínio: *PerformanceTest*

Imagem: *PTGoal*

Significado: Contém a informação dos objetivos que um teste de desempenho possui.

Propriedade: *hasLicense* (inversa de *isLicenseOf*)

Domínio: *PTTool*

Imagem: *SoftwareLicenseValue*

Significado: Mapeia as ferramentas de teste em uma instância de *SoftwareLicenseValue*.

Propriedade: *hostsApplication* (inversa de *isHostedOn*)

Domínio: *OperatingSystem*

Imagem: *ApplicationUnderTest*

Significado: Define o sistema operacional onde a aplicação sob teste está hospedada.

Propriedade: *imposesActivity* (inversa de *isImposedByGoal*)

Domínio: *PTGoal*

Imagem: *PTActivity*

Significado: Especifica quais atividades devem ser executadas para atingir um objetivo.

Propriedade: *measures* (inversa de *isMeasurementOf*)

Domínio: *PerformanceMeasurement*

Imagem: *PerformanceMetric*

Significado: Mapeia as medições das métricas de desempenho.

Propriedade: *monitors* (inversa de *isMonitoredBy*)

Domínio: *PerformanceTest*

Imagem: *PerformanceMetric*

Significado: Representa a informação das métricas de desempenho que um determinado teste é responsável por monitorar.

Propriedade: *supports* (inversa de *isSupportedBy*)

Domínio: *PTTool*

Imagem: *PTActivity*

Significado: Indica em quais atividades cada ferramenta de teste pode ser aplicada.

Propriedade: *testsApplication* (inversa de *isTestedBy*)

Domínio: *PerformanceTest*

Imagem: *ApplicationUnderTest*

Significado: Contém a informação de qual aplicação está sendo testada por uma instância de teste de desempenho.

Propriedade: *testsUseCase* (inversa de *isTestedByTestCase*)

Domínio: *PTCase*

Imagem: *ApplicationUseCase*

Significado: Representa qual caso de uso da aplicação é testado por cada caso de teste.

Propriedade: *usesProtocol* (inversa de *isUsedByApplicationUseCase*)

Domínio: *ApplicationUseCase*

Imagem: *CommunicationProtocol*

Significado: Mapeia quais protocolos de comunicação são utilizados em cada caso de uso da aplicação.

Propriedade: *usesTool* (inversa de *isUsedByTest*)

Domínio: *PerformanceTest*

Imagem: *PTTool*

Significado: Indica as ferramentas de teste escolhidas para um teste de desempenho.

A Figura 30 conclui esta seção e utiliza um diagrama de classes UML para resumir os principais termos representados na ontologia. Cada classe do diagrama simboliza um conceito e cada associação simboliza uma propriedade. Além disso, a origem da associação é o domínio da propriedade em OWL, enquanto que o destino é a imagem.

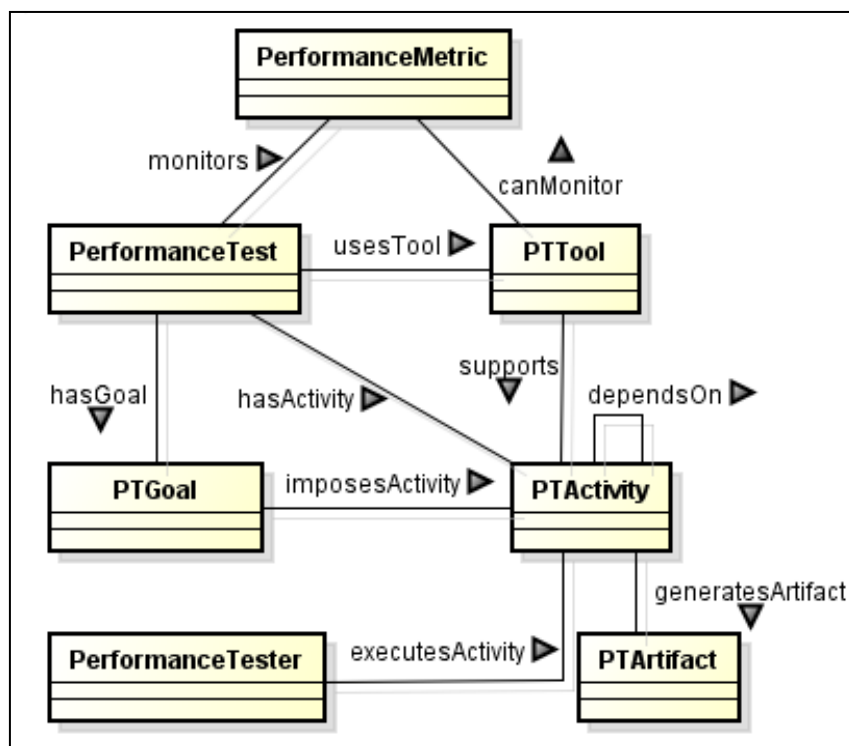


Figura 30: Diagrama de classe resumindo os principais conceitos e relacionamentos da ontologia.

5. AVALIAÇÃO DA ONTOLOGIA

"Not everything that can be counted counts and not everything that counts can be counted." (Albert Einstein)

Avaliar a qualidade de uma ontologia permite a identificação de partes que devem ser melhor especificadas e possibilita a comparação entre duas ou mais ontologias [64]. A qualidade de uma ontologia pode ser avaliada em diferentes dimensões, e.g., a nível estrutural ou a nível semântico [23]. Estas dimensões estão listadas a seguir [32]:

- **Léxico, vocabulário ou camada de dados:** foca nos conceitos, instâncias e fatos.
- **Hierarquia ou taxonomia:** ênfase na avaliação dos relacionamentos de subsunção.
- **Outras relações semânticas:** avalia outros relacionamentos da ontologia, o que pode incluir o cálculo de métricas como precisão e revocação (*precision and recall*).
- **Nível de contexto:** avalia como o uso de uma ontologia por uma aplicação afeta seus resultados do ponto de vista dos usuários da aplicação.
- **Nível sintático:** avalia se a ontologia (representada em uma linguagem formal) está de acordo com os requisitos sintáticos da linguagem.
- **Estrutura, arquitetura, projeto:** avalia questões estruturais como a organização da ontologia e sua adequação para desenvolvimentos posteriores.

Para avaliar uma ontologia nos níveis listados anteriormente, existem abordagens que podem ser classificadas de acordo com uma das seguintes categorias [32]:

- Comparar a ontologia com um padrão de ouro (*gold standard*);
- Usar a ontologia em uma aplicação e analisar os resultados;
- Comparar a ontologia com uma fonte de dados do domínio;
- Avaliar, por meio de especialistas do domínio, se a ontologia atende determinados critérios, padrões ou requisitos.

A importância de uma ontologia pode ser atribuída a razões como adequação a uma finalidade, capacidade de responder questões de competência, presença de propriedades formais, entre outros fatores [34]. Para uma ontologia ser uma boa especificação de um domínio, tanto a conceitualização do domínio quanto sua formalização e especificação precisam ser consideradas boas [34]. Estes dois requisitos estão ilustrados como (1) e (2) na Figura 31. A conceitualização (equivalente ao modelo mental do domínio) é resultado de processos cognitivos como percepção e reconhecimento [34]. Se a conceitualização for especificada em uma ontologia formal, então será possível avaliar sua qualidade [34].

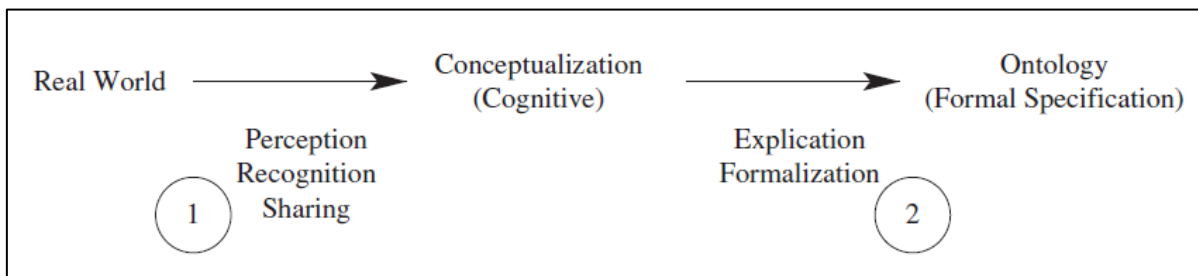


Figura 31: Qualidade das ontologias (Fonte [34]).

De acordo com [7], nenhuma abordagem completa para a avaliação de ontologias foi proposta até o momento. Segundo [31], a avaliação de ontologias permanece um importante problema em aberto. Apesar deste reconhecimento dos autores de que não existem técnicas perfeitas para a avaliação de ontologias, este trabalho aplicou duas das técnicas de avaliação de ontologias existentes, como será explicado nas subseções seguintes. As abordagens adotadas para avaliar a ontologia proposta foram: a comparação com duas ontologias de domínios semelhantes (OntoTest [19] e a SwTO [13]) e a avaliação da ontologia por especialistas no domínio de teste de desempenho.

5.1. Comparação com Ontologias Relacionadas

A comparação da qualidade de ontologias é especialmente vantajosa quando as ontologias comparadas cobrem o mesmo domínio [34]. Entretanto, como não foram encontradas ontologias sobre o domínio de teste de desempenho ou que possuam questões de competência semelhantes, a ontologia proposta foi comparada com duas ontologias relacionadas: a OntoTest [19] e a SwTO [13]. Contudo, é importante destacar que o domínio da ontologia proposta é o teste de desempenho, enquanto que as outras duas ontologias foram desenvolvidas para o domínio de teste de software.

A OntoTest [19] foi construída para apoiar a aquisição, compartilhamento, reuso e organização do conhecimento em teste de software. A OntoTest [19] explora os aspectos envolvidos na atividade de teste de software, define um vocabulário comum e permite estabelecer arquiteturas de referência, responsáveis por facilitar o uso e a integração de ferramentas, processos e artefatos na engenharia de software. A OntoTest foi usada para definir a RefTEST [19], uma arquitetura de referência sobre teste de software. Estabelecer estas arquiteturas exige conhecimento profundo do domínio, mas uma vez estabelecidas, estes conhecimentos permanecem associados em suas atividades e relacionamentos.

A segunda ontologia utilizada para comparação, denominada SwTO (Software Test Ontology) [13], é uma ontologia sobre o domínio de teste de software projetada para

auxiliar o teste no sistema operacional Linux. A SwTO foi utilizada para gerar sequências de teste para o Linux e, segundo os autores [13], três vantagens foram identificadas: a definição de um vocabulário formal sobre o domínio de teste no Linux que representa o consenso do grupo; o armazenamento semântico do critério de elaboração dos testes; e a representação semântica do conhecimento dos projetistas de teste. Mais informações sobre a OntoTest e a SwTO podem ser encontradas nas referências citadas neste trabalho e na seção 3 denominada Trabalhos Relacionados.

Uma vez que as três ontologias estão corretas a nível sintático (de acordo com a verificação executada em <http://owl.cs.manchester.ac.uk/validator/>), a Tabela 2 compara como cada uma das ontologias aborda os principais conceitos da área. Ontologias podem ser avaliadas e comparadas quantitativamente por meio de técnicas baseadas em métricas [63]. Este tipo de técnica descreve determinados aspectos da ontologia, ao invés de classificá-la como eficaz ou ineficaz [64]. Do ponto de vista do desenvolvedor de ontologias, estas métricas podem indicar áreas que necessitam de mais trabalho [23]. Por exemplo, a Tabela 2 ilustra diferenças quanto ao número de subclasses, instâncias e propriedades relacionadas aos principais conceitos destas ontologias.

Tabela 2: Comparação entre os conceitos da ontologia proposta com a OntoTest [19] e a SwTO [13].

Conceito	Comparação	Ontologia proposta	OntoTest [19]	SwTO [13]
Atividades de teste	Subclasses	29	20	7
	Instâncias	6	0	0
	Domínio de propriedades	6	0	1
Documentos do teste	Subclasses	10	8	4
	Instâncias	0	0	0
	Domínio de propriedades	1	3	1
Objetivos de teste	Subclasses	7	Conceito inexistente.	11
	Instâncias	0		0
	Domínio de propriedades	2		2
Ferramentas de teste	Subclasses	10	0	1
	Instâncias	18	1	32
	Domínio de propriedades	7	0	2
Artefatos gerados nos testes	Subclasses	13	7	Conceito inexistente.
	Instâncias	0	1	
	Domínio de propriedades	1	3	
Métricas de desempenho	Subclasses	23	Conceito inexistente.	Conceito inexistente.
	Instâncias	107		
	Domínio de propriedades	4		

Embora a Tabela 2 apresente uma forma de comparação entre diferentes ontologias, existem diferenças que não se limitam apenas a estas métricas. Em outras palavras, mesmo que duas ontologias apresentem um conceito considerado idêntico e que possua o mesmo número de subclasses nas duas ontologias, ainda assim não é possível afirmar que estas subclasses representam as mesmas entidades. Na SwTO [13], o conceito atividade de teste é chamado de *TestActivity* e é utilizado como domínio apenas da propriedade *isTestActivity*. Este mesmo conceito, na OntoTest [19], é chamado de *TestingActivity* e não foi definido como domínio de nenhuma propriedade. Por outro lado, na ontologia proposta, o conceito que representa uma atividade de teste (*PTActivity*) é usado como domínio de 6 diferentes propriedades: *dependsOn*, *generatesArtifact*, *isExecutedBy*, *isImposedByGoal*, *isPartOfTest* e *isSupportedBy*. A Tabela 2 mostra que a ontologia sobre teste de desempenho possui mais conceitos, que estes conceitos são utilizados em mais propriedades e que existem mais instâncias quando comparadas as outras duas ontologias. Isto é um indicador de que a ontologia proposta representa uma quantidade maior de informação.

A Tabela 3 ilustra métricas calculadas no software Protégé [62] para comparação da ontologia proposta com a OntoTest [19] e a SwTO [13]. Observando a Tabela 3, é possível constatar que a ontologia proposta possui 99 conceitos, se posicionando entre os 85 conceitos da SwTO e os 126 conceitos da OntoTest. A princípio, quanto maior o número de conceitos mais elementos do domínio estão representados na ontologia. Contudo, o número de conceitos em si não leva em consideração se estes conceitos estão corretos ou ainda se eles são úteis. De forma semelhante, a ontologia sobre teste de desempenho possui 44 propriedades relacionais, ficando entre as 19 propriedades da OntoTest e as 85 da SwTO. As propriedades são responsáveis por representar relações semânticas entre os conceitos de uma ontologia. Uma curiosidade que pode ser observada nesta comparação é que a OntoTest foi a ontologia com o maior número de conceitos, mas com o menor número de propriedades. Quanto ao quesito número de instância, a ontologia proposta possui 171 indivíduos, enquanto que a OntoTest possui 18 e a SwTO, 36. A maioria das instâncias na ontologia proposta pertence ao conceito métrica de desempenho ou ferramenta de teste, conforme apresentado na Seção 4. O número de axiomas lógicos da ontologia sobre teste de desempenho também é maior que o das outras duas ontologias: 3170 axiomas lógicos contra 295 (OntoTest) e 578 (SwTO). A explicação para esta grande diferença é que a ontologia proposta possui um número maior de instâncias, que por sua vez possuem relações entre si. Por exemplo, uma instância de ferramenta de teste se relaciona com diversas outras instâncias da ontologia,

como, por exemplo, métricas de desempenho, protocolos de comunicação, sistemas operacionais, etc. Em outras palavras, destes 3170 axiomas lógicos, 2506 são do tipo *ObjectPropertyAssertion*, sendo que as outras duas ontologias não possuem nenhum axioma deste tipo. Contudo, o número de axiomas *SubClassOf* na ontologia proposta, 118, é o menor valor quando comparado aos 161 da OntoTest e aos 166 da SwTO. A ontologia sobre teste de desempenho possui 231 axiomas do tipo *DisjointClasses* contra 180 da SwTO e nenhum da OntoTest. A OntoTest também não possui axiomas do tipo *AnnotationAssertion*, que são utilizados para anotar ou documentar os elementos da ontologia. A ontologia proposta apresenta 187 *AnnotationAssertion* e a SwTO possui 153 axiomas deste tipo. De acordo com [32], é importante que as definições formais e as declarações representadas na ontologia estejam acompanhadas de documentação em linguagem natural, sendo esta documentação significativa, coerente, atualizada e consistente com as definições formais.

Tabela 3: Métricas extraídas do Protégé para comparação das ontologias.

Métricas extraídas do Protégé: número de ...	Ontologia Proposta	OntoTest [19]	SwTO [13]
conceitos	99	126	85
propriedades relacionais	44	19	85
propriedades de dados	9	6	26
instâncias	171	18	36
axiomas lógicos	3170	295	576
axiomas <i>ObjectPropertyAssertion</i>	2506	0	0
axiomas <i>SubClassOf</i>	118	161	166
axiomas <i>DisjointClasses</i>	231	0	180
axiomas <i>AnnotationAssertion</i>	187	0	153
Expressividade em DL	SOIF(D)	SHOIQ(D)	SHI(D)

Além destas métricas, as ontologias podem ser comparadas quanto a expressividade em DL (Description Logic). As três ontologias apresentam relações de transitividade, como indicado pela letra S; propriedades inversas, como indica a letra I e utilizam tipos de dados primitivos, como indica a expressão (D). Apenas a ontologia proposta e a SwTO fazem uso de classes enumeradas ou restrições do tipo *oneOf* e *hasValue*, como indicado pela letra O. Além disso, a ontologia proposta é a única que utiliza propriedades funcionais, como indicado pela letra F; hierarquia de propriedades é

explorada apenas na OntoTest e na SwTO, como demonstra a letra H e, por fim, restrições qualificadas de cardinalidade são utilizadas apenas pela SwTO, como indica a letra Q. A expressividade em DL adotada pelo Protégé está descrita na Figura 32.

\mathcal{AL}	<p>Attributive language. This is the base language which allows:</p> <ul style="list-style-type: none"> ● Atomic negation (negation of concepts that do not appear on the left hand side of axioms) ● Concept intersection ● Universal restrictions ● Limited existential quantification (restrictions that only have fillers of Thing)
\mathcal{FL}^-	A sub-language of \mathcal{AL} , which is obtained by disallowing atomic negation
\mathcal{FL}_o	A sub-language of \mathcal{FL}^- , which is obtained by disallowing limited existential quantification
\mathcal{C}	Complex concept negation
\mathcal{S}	An abbreviation for \mathcal{AL} and \mathcal{C} with transitive properties
\mathcal{H}	Role hierarchy (subproperties - <code>rdfs:subPropertyOf</code>)
\mathcal{O}	Nominals. (Enumerated classes or object value restrictions - <code>owl:oneOf</code> , <code>owl:hasValue</code>)
\mathcal{I}	Inverse properties
\mathcal{N}	Cardinality restrictions (<code>owl:Cardinality</code> , <code>owl:minCardianlity</code> , <code>owl:maxCardinality</code>)
\mathcal{Q}	Qualified cardinality restrictions (available in OWL 1.1)
\mathcal{F}	Functional properties
\mathcal{E}	Full existential quantification (Existential restrictions that have fillers other that <code>owl:Thing</code>)
\mathcal{U}	Concept union
(\mathcal{D})	Use of datatype properties, data values or datatypes

Figura 32: Legenda utilizada para definir a expressividade em Description Logic adotada pelo Protégé.

Ontologias podem variar quanto à qualidade, cobertura e nível de detalhe [51]. Esta afirmação pode ser observada na prática segundo as comparações entre a ontologia proposta, a OntoTest [19] e a SwTO [13]. Ainda de acordo com [51], decidir se uma ontologia é apropriada para um uso em particular ainda é uma tarefa subjetiva. Isto se deve, principalmente, a escassez de métricas objetivas e computáveis para determinar a qualidade de uma ontologia [51]. A complexidade de avaliar e comparar ontologias se deve ao fato de que diferentes ontologias podem descrever as mesmas entidades utilizando diferentes conceitos ou, ainda, utilizar o mesmo conceito para representar diferentes entidades [43]. A justificativa para a ocorrência disto é que o conhecimento

pressupõe um grau de interpretação, resultando em diferentes perspectivas conhecidas como paradigmas, contextos, percursos cognitivos, espaços mentais, entre outros [43]. Portanto, uma ontologia é o resultado de um processo de interpretação (*sense-making*), que resulta em uma modelagem conceitual e que representa o ponto de vista de quem participou deste processo [43]. Apesar de tudo, estas comparações apresentadas são importantes para a identificação de aspectos como os levantados anteriormente. Possuir mais conceitos, propriedades, axiomas e instâncias podem indicar que o conhecimento está representado em um nível maior de detalhamento. Contudo, deve ser lembrado que a ontologia sobre teste de desempenho está sendo comparada com duas ontologias de domínios relacionados (teste de software), uma vez que não foi encontrada outra ontologia sobre teste de desempenho. Sabendo disto e após comparar a ontologia proposta com duas ontologias relacionadas, a próxima seção apresentará os resultados de entrevistas com especialistas em teste de desempenho visando avaliar a qualidade dos conceitos representados na ontologia desenvolvida.

5.2. Avaliação da Ontologia por Especialistas

A relação entre uma ontologia e uma conceitualização é sempre dependente da concepção desta conceitualização por um agente racional (semântica cognitiva) e da codificação formal desta conceitualização (semântica formal) [7]. Portanto, uma avaliação deve também incluir maneiras de medir o grau em que a ontologia reflete uma dada experiência [7], também conhecido como “o julgamento de especialistas”. O objetivo desta abordagem é medir o quanto os especialistas estão de acordo sobre os elementos existentes na ontologia. Assim, com a finalidade de avaliar a qualidade da ontologia, um questionário foi aplicado em especialistas em teste de desempenho (pessoas que possuem conhecimento teórico ou experiência prática na área). Em uma primeira etapa deste questionário foi solicitado para cada especialista enumerar o maior número de atividades, objetivos, métricas e ferramentas relacionadas ao domínio de teste de desempenho, uma vez que estes são os principais conceitos da ontologia. Na segunda etapa desta avaliação, foram apresentados os conceitos da ontologia para os especialistas que receberam a tarefa de avaliar o quanto estavam de acordo com a existência de cada conceito. Assim, é possível comparar as respostas dos especialistas com a representação da ontologia e identificar quais conceitos os especialistas conhecem, se eles concordam com os conceitos já existentes e também perguntar se existem novos conceitos que deveriam ser adicionados na ontologia. O questionário foi

respondido online² e pode ser encontrado no Apêndice A desta dissertação. As instruções gerais para todos que o responderam foram exatamente as seguintes:

“Você pode escolher responder as seguintes perguntas utilizando unicamente seus conhecimentos ou pode consultar qualquer site, livro ou outros materiais. Se algum material for consultado, é importante citar as fontes consultadas após cada resposta. Dado que as seguintes perguntas possuem mais de uma resposta, tente respondê-las listando a maior quantidade possível de respostas de acordo com o seu conhecimento. As respostas não possuem tamanho mínimo ou máximo, contudo, espera-se que quanto mais detalhadas as respostas melhores serão os resultados do questionário. Fica a critério do participante o nível de detalhamento das suas respostas. É importante que o participante não converse sobre este questionário ou suas respostas com nenhum outro participante que já respondeu ou irá responder este mesmo questionário.”

Em uma etapa de pré-questionário, os participantes foram perguntados sobre seus conhecimentos e experiências. Os 8 especialistas que responderam o questionário serão identificados pelos números de 1 a 8. A formação acadêmica dos participantes pode ser visualizada na Tabela 4 sendo todos os 8 ligados ao curso de Ciência da Computação: 3 atualmente cursando graduação, 2 cursando mestrado e 3 cursando doutorado. Além disso, quando perguntados há quanto tempo estudam ou trabalham na área de teste de desempenho, os participantes relataram que sua experiência na área foi adquirida em no mínimo 2 meses (participante 3) e em no máximo 4 anos (participante 5), sendo a média do tempo de experiência dos participantes em torno de 19 meses. As respostas completas dos 8 especialistas podem ser encontradas no Apêndice B desta dissertação.

Tabela 4: Formação acadêmica dos especialistas entrevistados.

Identificação do especialista	Formação acadêmica	Tempo de experiência em teste
1	Mestrado em andamento	7 meses
2	Doutorado em andamento	2 anos e meio
3	Graduação em andamento	2 meses
4	Graduação em andamento	6 meses
5	Doutorado em andamento	4 anos
6	Graduação em andamento	1 ano
7	Mestrado em andamento	2 anos
8	Doutorado em andamento	2 anos e meio
Média		19.87 meses

² O questionário foi hospedado em <http://www.surveymzmo.com/>.

Em seguida, os participantes foram perguntados sobre quais cargos profissionais ou acadêmicos relacionados ao teste de desempenho eles já ocuparam ou ainda ocupam. Resumidamente, 3 participantes adquiriram suas experiências em teste como estagiário, 3 participantes como bolsistas de mestrado e 2 como doutorandos. É importante destacar que todos os participantes estiveram ou ainda estão envolvidos no mesmo projeto de pesquisa sobre teste de desempenho resultante de um convênio da PUCRS e de uma grande empresa de TI.

Em seguida, os participantes foram questionados sobre onde e como adquiriram a maior parte de seus conhecimentos em teste de desempenho, e também, se eles classificariam seus conhecimentos mais como práticos ou teóricos. Apenas 2 participantes classificaram seus conhecimentos mais como teóricos, 1 participante definiu seus conhecimentos mais como práticos e os demais 5 participantes responderam que seus conhecimentos práticos e teóricos estão na mesma proporção.

O questionário foi dividido em uma parte inicial de perguntas abertas e uma parte final de perguntas fechadas. Os participantes responderam as perguntas abertas ainda sem conhecer os conceitos, propriedades e instâncias da ontologia. A finalidade das perguntas abertas é verificar os conhecimentos dos especialistas e comparar suas respostas com a ontologia. Após responder as perguntas abertas, os especialistas foram apresentados aos conceitos da ontologia por meio de perguntas fechadas que visavam avaliar o quanto eles estão de acordo com um determinado conceito. As perguntas fechadas do questionário seguiram a escala Likert [38], que é uma das escalas mais adotadas para pesquisas de opinião. De acordo com esta escala, os participantes devem escolher a alternativa mais adequada entre as seguintes opções:

- 1 – Discordo totalmente
- 2 – Discordo
- 3 – Não discordo e nem concordo
- 4 – Concordo
- 5 – Concordo totalmente

A primeira pergunta sobre teste que os participantes tiveram que responder foi “Quais objetivos um teste de desempenho pode possuir?”. De acordo com a ontologia proposta existem 7 conceitos que representam os objetivos de teste, contudo os participantes ainda não têm conhecimento desta ontologia. As pessoas questionadas enumeram no mínimo 1 e no máximo 4 objetivos dos testes de desempenho (em média 1.75). O Apêndice B apresenta todas as respostas do questionário e o Apêndice C compara estas respostas com a ontologia. O maior consenso (4 votos de 8) é que um dos

objetivos do teste de desempenho é a “avaliação do desempenho sob condições normais de carga”. Além disso, apenas um dos objetivos identificados pelos especialistas não está representado na ontologia, que é “estimar uma configuração de hardware adequada para cada aplicação”. Dos conceitos relacionado aos objetivos de teste, o que obteve a maior média de concordância (4.62) foi a “avaliação do desempenho durante longos períodos de interação” e a pior média (3.75) se refere a “elaboração de um entregável de teste de desempenho”. Na escala adotada o valor 4 simboliza que a pessoa concorda com afirmação em questão.

Também faz parte do questionário a seguinte pergunta: “Quais ferramentas existem para teste de desempenho e o que você sabe sobre elas?”. Em média, os participantes enumeraram 2.87 ferramentas (no mínimo 1 e no máximo 6). A ontologia tem informação sobre 18 instâncias do conceito *PTTool*, contudo, apenas 4 ferramentas foram citadas pelos participantes. O Visual Studio e o LoadRunner foram as ferramentas mais citadas, por 7 dos 8 participantes. Em segundo lugar, o JMeter foi citado por 5 especialistas.

O questionário solicitava também a enumeração das atividades que podem fazer parte de um teste de desempenho. Em média, os participantes enumeraram 6.87 atividades, das quais 6.25 já estão representadas na ontologia (que possui 29 conceitos especializando *PTActivity*). Todos os 8 participantes foram unânimes em responder as atividades de “análise das medições de desempenho” e “execução dos casos de teste”. Em segundo lugar, 6 participantes citaram as atividades de “desenvolvimento dos casos de teste” e “elaboração do plano de teste”. Contudo, algumas atividades não foram lembradas por nenhum participante, como, por exemplo, a “parametrização dos casos de teste de desempenho”. Além disso, existem atividades citadas que atualmente não fazem parte da ontologia, como a “modelagem do teste de desempenho”, caso seja utilizada uma abordagem baseada em modelos para geração do teste. Quando apresentado aos conceitos do tipo *PTActivity*, a maior média de aceitação (4.25) ficou com a atividade “monitoramento das métricas de desempenho”. Ainda, as atividades foram agrupadas em planejamento, desenvolvimento, execução e documentação. Quando perguntados o quanto os especialistas estavam de acordo com estas categorias as respostas foram: execução em primeiro lugar com média de 4.62, planejamento e desenvolvimento empatadas com 4.5 e documentação com 4.37 de média. Ainda, 2 participantes disseram que adicionariam uma nova categoria na ontologia denominada “atividades de análise”.

Faz parte do questionário também a pergunta “Quais artefatos (ou entregáveis) podem resultar de um teste de desempenho?”. Embora os participantes enumeraram em

média 2.62 entregáveis, a ontologia possui 13 conceitos especializando *PTArtifact*. Ao total, os participantes citaram 6 diferentes conceitos do tipo *PTArtifact*, sendo o “relatório do teste de desempenho” o mais citado, por 7 dos 8 questionados. Foi lembrado também que o caso de teste, além de ser um script para uma ferramenta de teste, pode ser um modelo caso o testador use uma abordagem baseada em modelos. Quando apresentado aos conceitos da ontologia do tipo *PTArtifact*, o “relatório do teste de desempenho” recebeu a maior média (4.25). Este tinha sido o entregável mais citado pelos especialistas antes deles conhecerem os conceitos da ontologia.

Uma outra pergunta do questionário foi a seguinte: “Quais métricas de desempenho podem ser monitoradas?”. Em média os participantes enumeraram 11.75 diferentes respostas para essa questão, das quais 10.12 já estão representadas na ontologia como conceitos ou instâncias do conceito *PerformanceMetric*. A ontologia possui 107 instâncias e 23 subclasses que especializam este conceito. Além disso, todos os 8 participantes citaram *MachinePerformanceMetric*, 7 identificaram a instância que representa a métrica “tempo de resposta” e 6 responderam o conceito *PTMetric*.

A Tabela 5 compara quantas respostas foram dadas pelos especialistas, em média, e quantos conceitos e instâncias a ontologia possui para representar os principais elementos do domínio de teste de desempenho. Os valores desta tabela indicam que a ontologia pode representar e tornar acessível uma quantidade maior de informações quando comparada as respostas dos especialistas. Além disso, a coluna mais direita da Tabela 5 apresenta a porcentagem de semelhança das respostas dos especialistas com relação ao número de total de respostas. Em outras palavras, a porcentagem de semelhança é o resultado da divisão do número de respostas semelhantes pelo número total de respostas. Por exemplo, ao total, das 55 atividades de teste identificadas pelos especialistas, 50 já estão representadas como conceitos da ontologia, resultando em uma porcentagem de semelhança de 90.91% (50 dividido por 55). Maiores detalhes sobre o questionário, as respostas e a comparação com a ontologia podem ser encontrados, respectivamente, nos Apêndices A, B e C da dissertação. Como mostra a Tabela 5, as respostas dos especialistas são semelhantes aos elementos da ontologia, embora eles não tenham identificado tantos elementos quanto a ontologia possui. O menor valor quanto a porcentagem de semelhança foi apresentado pelos conceitos das ferramentas de teste, com 46.67%. É importante observar que as semelhanças mostram que o domínio está bem conceitualizado, contudo, as diferenças são uma oportunidade para aumentar o nível de detalhamento da ontologia por meio da adição de novos conceitos,

propriedades ou indivíduos. Os conceitos identificados pelos especialistas que atualmente não fazem parte da ontologia foram os seguintes:

- **Ferramenta para geração de teste.** Este conceito pode ser criado como qualquer ferramenta de teste (subclasse de *PTTool*) que suporta (propriedade *supports*) a atividade de desenvolvimento dos casos de teste (conceito *PTCaseDevelopment*).
- **Ferramenta de análise.** Este tipo de ferramenta deve suportar as atividades de análise das medições de desempenho (*PerformanceMeasurementAnalysis*) e elaboração do relatório de teste (*PTReportElaboration*).
- **Ferramenta open source.** As ferramentas open source são instâncias de *PTTool* que têm como licença de uso (propriedade *hasLicense*) a instância *OpenSourceLicense*.
- **Ferramenta proprietária.** Semelhante ao conceito anterior, são instâncias de *PTTool* que se relacionam pela propriedade *hasLicense* com a instância *ProprietaryLicense*.

Tabela 5: Comparação entre as respostas dos especialistas e os elementos da ontologia.

		Média de respostas dos especialistas	Elementos na ontologia	Porcentagem de semelhança
Objetivos		1.75	7	92.86 %
Atividades	Conceitos	6.87	29	90.91 %
Entregáveis		2.62	13	76.19 %
Ferramentas	Conceitos	1.88	10	46.67 %
	Instâncias	2.87	18	86.95 %
Métricas	Conceitos	4.50	23	91.67 %
	Instâncias	7.25	107	82.76 %

A Tabela 6 mostra a avaliação geral dos especialistas com relação aos conceitos da ontologia que lhes foram apresentados. Ao total, cada um dos 36 conceitos foram avaliados pelos 8 especialistas, resultando em 288 respostas. Dentro da escala Likert adotada (de 1 a 5), a média de respostas ficou em 4.03, lembrando que o valor 4 significa que o especialista está de acordo com o conceito apresentado. Os conceitos que obtiveram o maior nível de aceitação foram os do tipo *PTGoal*, com média de 4.16. Entretanto, os conceitos do tipo *PTArtifact* obtiveram a média mais baixa de aceitação, com 3.85. Com base nisso, podem ser levantadas as hipóteses de que a conceitualização

não reflete com precisão o conhecimento do domínio ou que os especialistas não estão habituados com os conceitos apresentados. Como as subclasses de *PTArtifact* foram extraídas do padrão IEEE 829 [27], ou seja, fazem parte do conhecimento do domínio, o mais provável é que os especialistas não estivessem familiarizados com estes termos. Neste caso, a ontologia pode ajudá-los a conhecer ainda melhor os principais conceitos do domínio de teste de desempenho.

Tabela 6: Avaliação geral dos especialistas sobre os conceitos da ontologia.

Conceitos apresentados	Número de conceitos	Avaliação dos especialistas (escala Likert de 1 a 5)
Objetivos (<i>PTGoal</i>)	7	4.16
Atividades (<i>PTActivity</i>)	20	4.07
Entregáveis (<i>PTArtifact</i>)	9	3.85
Total	36	4.03

Com base na avaliação dos especialistas é possível concluir que, como mostra a Tabela 5, a ontologia representa e torna acessível mais informações sobre o domínio do que os especialistas conseguiram responder. Além disso, de acordo com a Tabela 6, os especialistas mostraram estar de acordo com os elementos representados na ontologia, apesar destes elementos não estarem presentes em suas respostas anteriores.

É interessante observar no Apêndice B que muitas das referências citadas pelos especialistas para responder as perguntas também haviam sido utilizadas para definir a ontologia, estando presentes na seção de referências desta dissertação. Além disso, foi comentado por dois especialistas que a ontologia representa o teste de desempenho de maneira convencional, não distinguindo o fato de que o teste de desempenho possa ser baseado em modelos. Seria possível expandir os conceitos da ontologia neste sentido, de modo a cobrir o domínio de teste de desempenho baseado em modelos.

Esta seção contemplou os resultados da comparação da ontologia proposta com outras duas ontologias relacionadas e, também, as opiniões de especialistas em teste de desempenho sobre a ontologia. A comparação com a OntoTest [19] e SwTO [13] permitiu identificar semelhanças e diferenças da ontologia proposta com relação a duas ontologias existentes. A avaliação por especialistas possibilitou semelhanças e diferenças quanto ao conhecimento deles sobre o domínio e o conhecimento representado na ontologia. A seção seguinte apresentará aplicações baseadas nesta ontologia.

6. APLICAÇÕES DA ONTOLOGIA PROPOSTA

“Test planning can be likened to a chess game. There are a number of pieces on your side of the board, each available to help you win the game. Your challenge is to develop a winning strategy and advance the specific chess pieces that will support your strategy.” [24]

Aplicações que utilizam a ontologia podem ser desenvolvidas de diferentes formas como, por exemplo, estendendo as funcionalidades do Protégé ou utilizando alguma API para manipulação da ontologia. Assim sendo, as seções seguintes exemplificam algumas destas maneiras de utilizar a ontologia proposta. Embora estas aplicações de ontologias utilizem abordagens variadas, o objetivo é sempre auxiliar o testador de software a estabelecer testes de desempenho, como apresentado na Seção "4.1 Escopo da Ontologia". O objetivo é apoiar as decisões dos testadores com o conhecimento das tecnologias utilizadas em testes de desempenho do software, validando e recomendando opções de acordo com o ambiente de teste, os objetivos e as atividades que o teste pode possuir. Para isto, a aplicação deve consultar a ontologia que possui conhecimento sobre as atividades necessárias para a realização de um teste de desempenho, as ferramentas que podem resolver tais atividades, quais são os requisitos para a utilização de uma dada ferramenta de teste, entre outros. Uma das premissas para as aplicações baseadas na ontologia é que a identificação de ferramentas, atividades, artefatos e métricas adequadas para um teste específico requer determinados conhecimentos prévios sobre o domínio. Neste sentido, as aplicações podem auxiliar os testadores de acordo com os conhecimentos da ontologia. Para executar inferências sobre o domínio, os axiomas existentes na ontologia são manipulados a fim de responder as perguntas elaboradas de acordo com o escopo da ontologia, que são as seguintes: De acordo com o ambiente de teste, quais ferramentas podem ser utilizadas? De acordo com o objetivo do teste, que atividades devem ser executadas? De acordo com os objetivos do teste e ambiente de teste, quais métricas de desempenho podem ser coletadas? Na seção 4, a Figura 15 apresentou os casos de uso que guiaram o desenvolvimento da ontologia. Portanto, as aplicações baseadas nesta ontologia deverão executar estes casos de uso.

6.1. Aplicação Desenvolvida como Plugin do Protégé

Uma das vantagens encontradas nas ontologias se deve a existência de ferramentas que apoiam seu desenvolvimento e utilização. O Protégé pode ser configurado para

permitir a exibição e manipulação da ontologia de forma intuitiva pelos usuários finais. Ainda, o Protégé possui uma arquitetura que permite o desenvolvimento de plugins que adicionem novas funcionalidades à ferramenta. Desta forma, é possível aproveitar as funcionalidades já existentes no Protégé, como por exemplo, a utilização de *reasoners* e a execução de regras escritas em SWRL e SQWRL. Foram desenvolvidos quatro novos componentes para o Protégé: dois *tab widget* e dois *slot widget* plugins. Os plugins do tipo *tab widget* são novas abas que podem aparecer na interface do Protégé e serem utilizadas para implementar novas funcionalidades. Em contrapartida, um plugin do tipo *slot widget* é um componente gráfico que estende a interface gráfica do Protégé com o objetivo de ampliar a visualização de informações e a edição de indivíduos e propriedades de uma ontologia. Os plugins *tab widget* foram denominados *ActivitiesDependencies* e *TestPlanGenerator*, como pode ser visualizado na Figura 33 (o Protégé pode ser configurado para exibir ou ocultar suas abas).

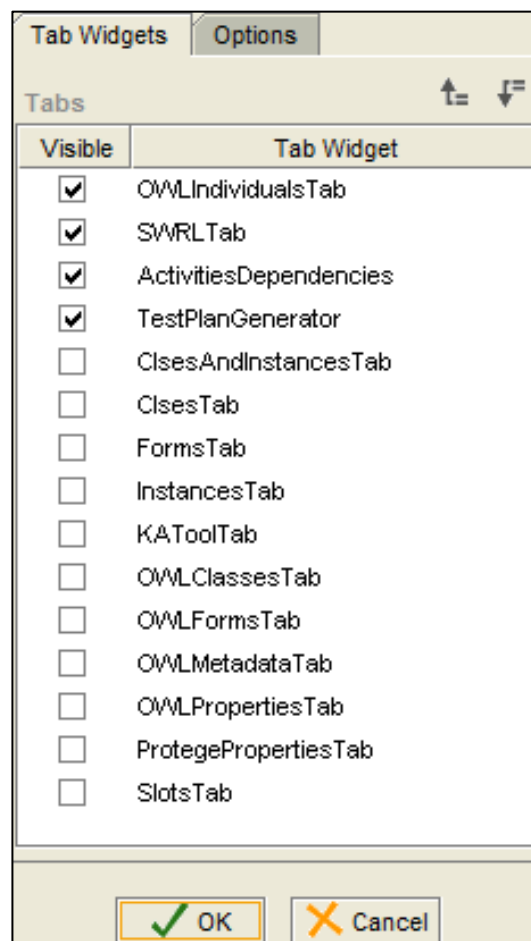


Figura 33: Personalização das abas em exibição na interface do Protégé.

O plugin *ActivitiesDependencies* é uma nova aba para o Protégé que infere quais atividades devem fazer parte de uma dada instância de teste de desempenho. Para fazer

isto, é levado em consideração que o conceito *PerformanceTest* é domínio da propriedade *hasActivity*, que tem como imagem *PTActivity*, indicando que um teste de desempenho pode possuir múltiplas atividades. Por sua vez, cada atividade pode apresentar a relação de dependência com outras *PTActivity*, por meio da propriedade *dependsOn*. Com estes conhecimentos, é gerado um grafo, como ilustrado na Figura 34, onde o nodo cinza indica a instância de teste de desempenho, os nodos verdes indicam as atividades que fazem parte deste teste e os nodos vermelhos indicam atividades que deveriam fazer parte do teste, porque são requisitos de alguma atividade existente. A visualização do grafo foi implementada em Java com apoio da JUNG [37] (Java Universal Network/Graph Framework). A JUNG é uma biblioteca *open source* para modelagem, análise e visualização de grafos ou redes.

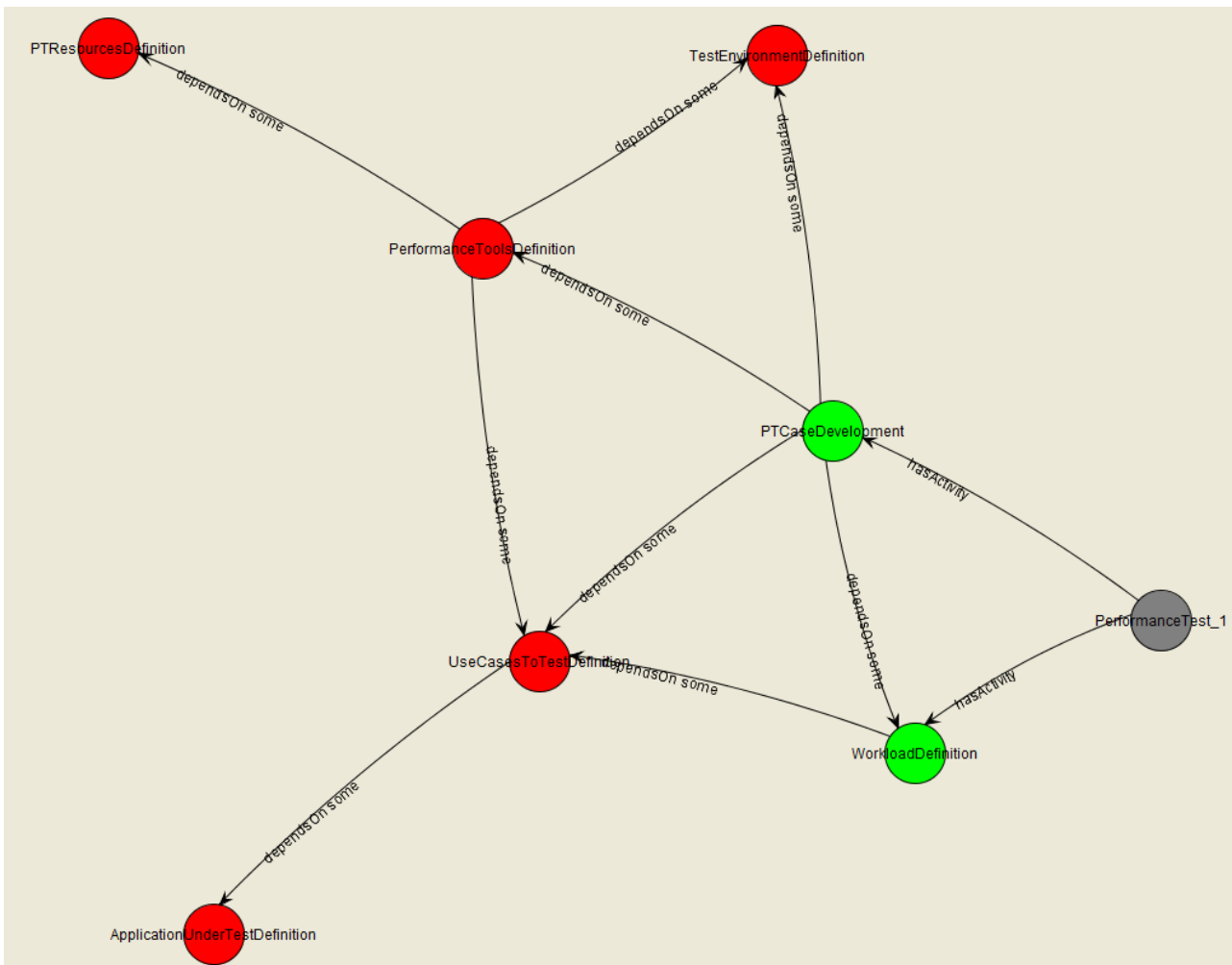


Figura 34: Exemplo de grafo de dependência de atividades gerado pelo plugin *ActivityDependencies*.

Os plugins do tipo *slot* foram denominados *AnnotationsWidget* e *RestrictionsWidget*. Estes dois componentes utilizam herança para especializar a classe *AbstractSlotWidget*

(disponível no pacote `edu.stanford.smi.protege.widget` da API do Protégé). Desta forma, é possível personalizar a interface original do Protégé e permitir a exibição de características não disponíveis em sua versão padrão [62]. Os componentes gráficos que estendem *SlotWidget* são apresentados para o usuário no momento da instanciação dos conceitos da ontologia. A Figura 35 demonstra o plugin *RestrictionsWidget* em uso na edição do formulário de instâncias do conceito *PerformanceTest*. O *RestrictionsWidget* permite a visualização das restrições representadas na ontologia para um determinado conceito. O componente *AnnotationsWidget* também está presente na Figura 35 (parte esquerda inferior), sendo responsável por exibir as anotações e comentários de um determinado conceito da ontologia. Esta parte da interface do Protégé corresponde às informações exibidas pela aba denominada *OWLFormsTab*.

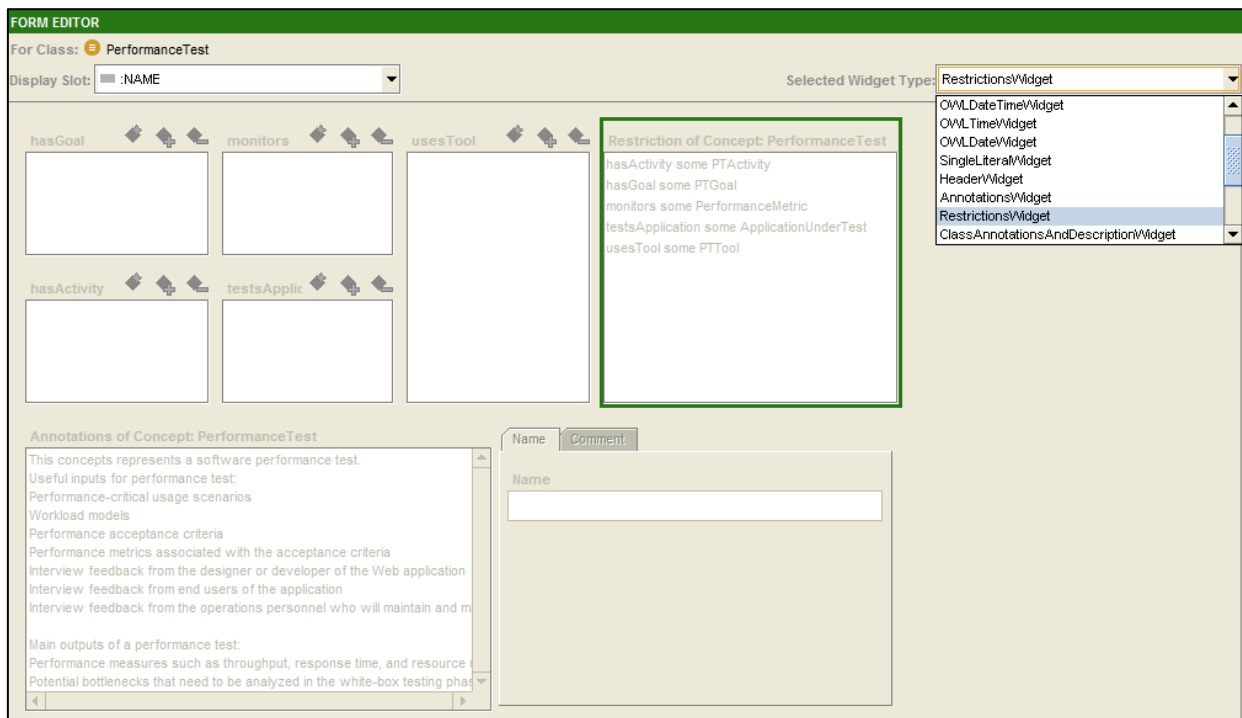


Figura 35: Demonstração do plugin *RestrictionsWidget* (painel com borda verde) sendo utilizado pelo Protégé.

Nesta abordagem de utilização da ontologia, as funcionalidades que o Protégé oferece podem ser aproveitadas. Um exemplo é a instanciação dos conceitos e propriedades, que é uma funcionalidade oferecida por uma aba já existente no Protégé, denominada *OWLIndividualsTab* ou apenas *Individuals*. A Figura 36 ilustra como criar um novo indivíduo do conceito *PerformanceTest*. Após criar uma nova instância de um conceito da ontologia, o resultado esperado é mostrado na Figura 37: no contexto denominado “Individual Editor” deve ser possível visualizar e editar informações relacionadas a instância selecionada. Observe também na Figura 37 que foram marcados

em vermelho que a instância possui inconsistências de acordo com as restrições do conceito ao qual ela pertence. Por exemplo, um conceito de *PerformanceTest* deve se relacionar com pelo menos uma instância de cada um dos seguintes conceitos: *PTGoal*, *PTActivity*, *PerformanceMetric*, *ApplicationUnderTest* e *PTTool* (segundo as restrições de *PerformanceTest*, apresentadas na Figura 17).

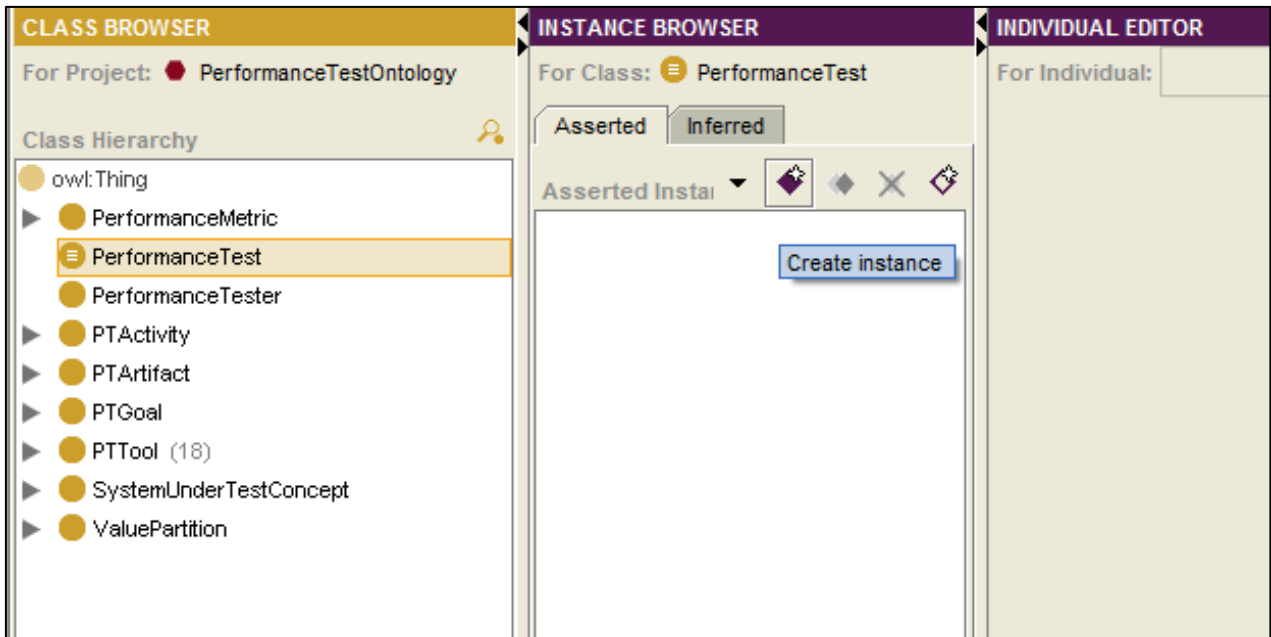


Figura 36: Criação de uma nova instância do conceito *PerformanceTest*.

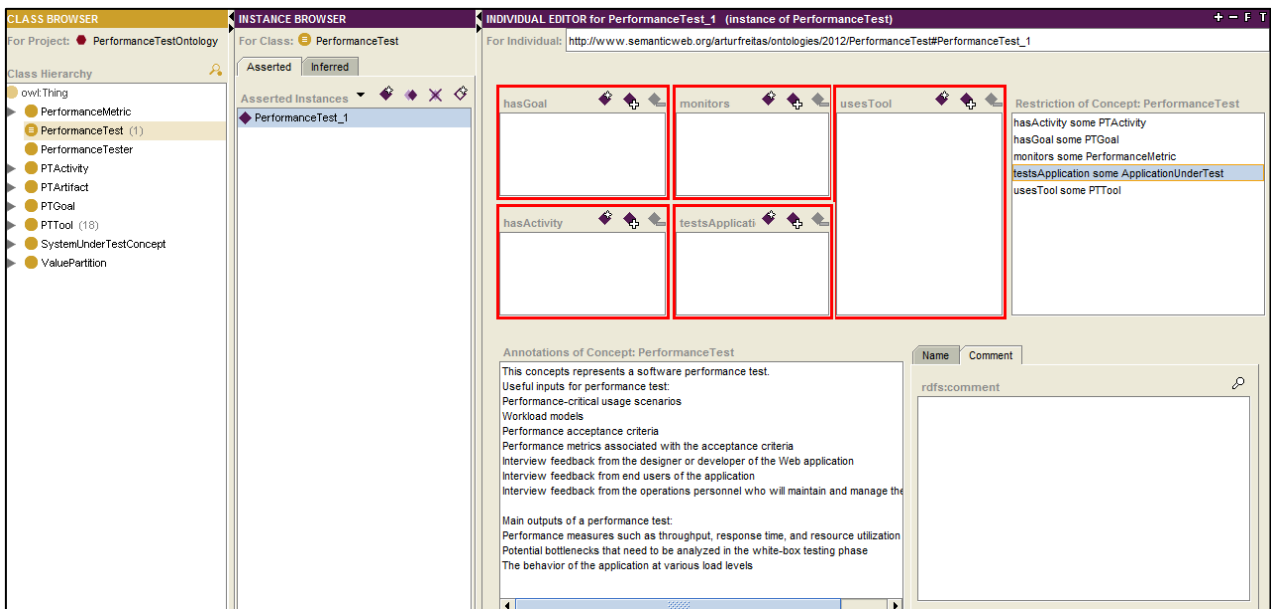


Figura 37: Edição de informações relacionadas a uma instância do conceito *PerformanceTest*.

As informações apresentadas ao usuário diferem de acordo com as propriedades de cada conceito sendo instanciado, de acordo com os axiomas da ontologia. Em outras

palavras, as informações variam de acordo com o conceito sendo instanciado, que pode ser *PerformanceTest*, *PTTool*, *PTActivity*, entre outros. Além de facilitar a instanciação, outra vantagem de utilizar o Protégé é a execução de consultas em SQWRL sobre a ontologia, como será apresentado na seção seguinte.

6.2. Execução de Consultas SQWRL sobre a Ontologia

Uma outra forma de explorar a ontologia proposta é utilizando consultas em SQWRL (Semantic Query-enhanced Web Rule Language). Segundo [47], SQWRL fornece uma linguagem simples e expressiva para executar consultas em ontologias OWL. Além disso, as consultas em SQWRL são serializadas juntamente com a ontologia. Ainda de acordo com [47], uma implementação *open source* e gratuita destas funcionalidades está incluída no Protégé, e a utilização desta implementação foi explorada para a ontologia proposta. A Figura 38 demonstra os nomes, as expressões das regras implementadas em SQWRL e o resultado da consulta denominada “Overall Generating Load Tools”, que retorna quais protocolos de comunicação cada ferramenta de teste consegue gerar carga. Uma explicação mais detalhada das regras desenvolvidas está disponível a seguir:

Nome da regra: *Disk_Monitoring_Tools*

Resultado esperado: Retorna instâncias do conceito *PTTool* que podem monitorar pelo menos uma métrica do disco. Utiliza a propriedade *canBeCollectedByTool*.

Expressão lógica: $\text{DiskMetric}(?x) \wedge \text{canBeCollectedByTool}(?x, ?y) \rightarrow \text{sqwrl:select}(?y, ?x) \wedge \text{sqwrl:columnNames}(\text{"Performance Test Tool"}, \text{"Performance Metric"}) \wedge \text{sqwrl:orderBy}(?x)$

Nome da regra: *HTTP_Generating_Load_Tools*

Resultado esperado: Retorna instâncias do conceito *PTTool* que podem gerar carga no protocolo HTTP. Utiliza a propriedade *generatesLoadOn*.

Expressão lógica: $\text{PTTool}(?x) \wedge \text{generatesLoadOn}(?x, ?y) \wedge \text{sameAs}(?y, \text{HTTP}) \rightarrow \text{sqwrl:select}(?x, ?y) \wedge \text{sqwrl:columnNames}(\text{"Performance Test Tool"}, \text{"Communication Protocol"}) \wedge \text{sqwrl:orderBy}(?y)$

Nome da regra: *Linux_Monitoring_Tools*

Resultado esperado: Retorna instâncias do conceito *PTTool* que podem monitorar pelo menos um sistema operacional baseado no Linux. Utiliza a propriedade *canMonitor*.

Expressão lógica: $\text{PTTool}(?x) \wedge \text{canMonitor}(?x, ?y) \wedge \text{LinuxBasedOperatingSystem}(?y) \rightarrow \text{sqwrl:select}(?x, ?y) \wedge \text{sqwrl:columnNames}(\text{"Performance Test Tool"}, \text{"Operational System"}) \wedge \text{sqwrl:orderBy}(?x)$

Nome da regra: *Measurable_Performance_Metrics*

Resultado esperado: Retorna instâncias do conceito *PTTool* que podem monitorar pelo menos uma métrica de desempenho. Utiliza a propriedade *canBeCollectedByTool*.

Expressão lógica: $\text{PerformanceMetric}(?x) \wedge \text{canBeCollectedByTool}(?x, ?y) \rightarrow \text{sqwrl:select}(?y, ?x) \wedge \text{sqwrl:columnNames}(\text{"Performance Test Tool"}, \text{"Performance Metric"}) \wedge \text{sqwrl:orderBy}(?x)$

Nome da regra: *Memory_Monitoring_Tools*

Resultado esperado: Retorna instâncias do conceito *PTTool* que podem monitorar pelo menos uma métrica de memória. Utiliza a propriedade *canBeCollectedByTool*.

Expressão lógica: $\text{MemoryMetric}(?x) \wedge \text{canBeCollectedByTool}(?x, ?y) \rightarrow \text{sqwrl:select}(?y, ?x) \wedge \text{sqwrl:columnNames}(\text{"Performance Test Tool"}, \text{"Performance Metric"}) \wedge \text{sqwrl:orderBy}(?x)$

Nome da regra: *Network_Monitoring_Tools*

Resultado esperado: Retorna instâncias do conceito *PTTool* que podem monitorar pelo menos uma métrica de rede. Utiliza a propriedade *canBeCollectedByTool*.

Expressão lógica: $\text{NetworkMetric}(?x) \wedge \text{canBeCollectedByTool}(?x, ?y) \rightarrow \text{sqwrl:select}(?y, ?x) \wedge \text{sqwrl:columnNames}(\text{"Performance Test Tool"}, \text{"Performance Metric"}) \wedge \text{sqwrl:orderBy}(?x)$

Nome da regra: *Overall_Generating_Load_Tools*

Resultado esperado: Retorna instâncias do conceito *PTTool* que podem gerar carga em pelo menos um protocolo de comunicação. Utiliza a propriedade *generatesLoadOn*.

Expressão lógica: $\text{PTTool}(?x) \wedge \text{generatesLoadOn}(?x, ?y) \rightarrow \text{sqwrl:select}(?x, ?y) \wedge \text{sqwrl:columnNames}(\text{"Performance Test Tool"}, \text{"Communication Protocol"}) \wedge \text{sqwrl:orderBy}(?y)$

Nome da regra: *Overall_Monitoring_Tools*

Resultado esperado: Retorna instâncias do conceito *PTTool* que podem monitorar pelo menos um sistema operacional. Utiliza a propriedade *canMonitor*.

Expressão lógica: $\text{PTTool}(?x) \wedge \text{canMonitor}(?x, ?y) \rightarrow \text{sqwrl:select}(?x, ?y) \wedge \text{sqwrl:columnNames}(\text{"Performance Test Tool"}, \text{"Operational System"}) \wedge \text{sqwrl:orderBy}(?x)$

Nome da regra: *Process_Monitoring_Tools*

Resultado esperado: Retorna instâncias do conceito *PTTool* que podem monitorar pelo menos uma métrica de processo. Utiliza a propriedade *canBeCollectedByTool*.

Expressão lógica: $\text{ProcessMetric}(?x) \wedge \text{canBeCollectedByTool}(?x, ?y) \rightarrow \text{sqwrl:select}(?y, ?x) \wedge \text{sqwrl:columnNames}(\text{"Performance Test Tool"}, \text{"Performance Metric"}) \wedge \text{sqwrl:orderBy}(?x)$

Nome da regra: *SMTP_Generating_Load_Tools*

Resultado esperado: Retorna instâncias do conceito *PTTool* que podem gerar carga no protocolo SMTP. Utiliza a propriedade *generatesLoadOn*.

Expressão lógica: $\text{PTTool}(?x) \wedge \text{generatesLoadOn}(?x, ?y) \wedge \text{sameAs}(?y, \text{SMTP}) \rightarrow \text{sqwrl:select}(?x, ?y) \wedge \text{sqwrl:columnNames}(\text{"Performance Test Tool"}, \text{"Communication Protocol"}) \wedge \text{sqwrl:orderBy}(?y)$

Nome da regra: *Windows_Monitoring_Tools*

Resultado esperado: Retorna instâncias do conceito *PTTool* que podem monitorar pelo menos um sistema operacional baseado no Windows. Utiliza a propriedade *canMonitor*.

Expressão lógica: $\text{PTTool}(?x) \wedge \text{canMonitor}(?x, ?y) \wedge \text{WindowsBasedOperatingSystem}(?y) \rightarrow \text{sqwrl:select}(?x, ?y) \wedge \text{sqwrl:columnNames}(\text{"Performance Test Tool"}, \text{"Operational System"}) \wedge \text{sqwrl:orderBy}(?x)$

The screenshot shows the SWRL Rules editor interface. At the top, there is a list of rules with columns for 'Enabled', 'Name', and 'Expression'. The rule 'Overall_Generating_Load_Tools' is selected and highlighted in blue. Below the list, there are tabs for 'SQWRLQueryTab', 'OWL 2 RL', and 'Overall_Generating_Load_Tools'. The main area displays a query result table with two columns: 'Performance Test Tool' and 'Communication Protocol'. The table contains the following data:

Performance Test Tool	Communication Protocol
SOAtest	HTTP
NeoLoad	HTTP
Ostinato	ICMPv4
Ostinato	ICMPv6
JMeter	IMAP
LoadRunner	IMAP
mstone	IMAP
Clif	IMAP
Ostinato	IPv4
Ostinato	IPv6

At the bottom of the window, there are buttons for 'Save as CSV...', 'Rerun', and 'Close'.

Figura 38: Exemplo de consultas em SQWRL sobre a ontologia de teste de desempenho.

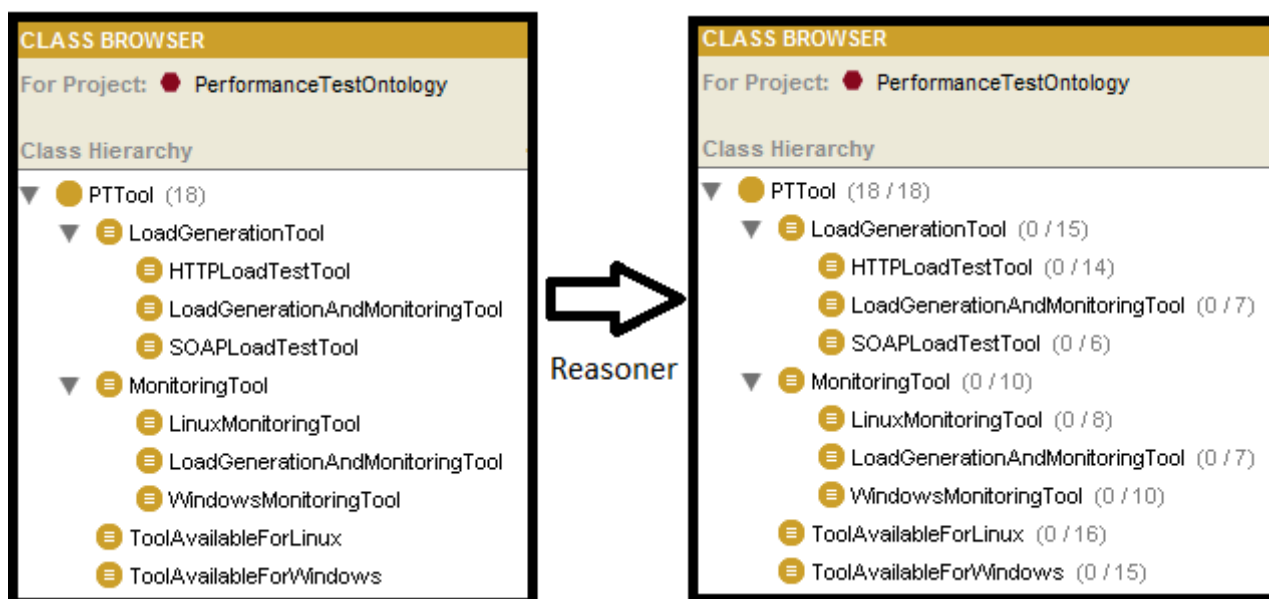
6.3. Utilização de Reasoners sobre a Ontologia

É esperado que um *reasoner* OWL apresente as funcionalidades de verificação de consistência, satisfatibilidade dos conceitos, classificação e realização [21]. A verificação de consistências garante que a ontologia não contém fatos contraditórios; a satisfatibilidade dos conceitos verifica se é possível para um conceito possuir alguma instância; a classificação computa relações hierárquicas entre as classes; e realização encontra conceitos aos quais os indivíduos da ontologia pertencem [21]. Em outras palavras, *reasoners* são capazes de inferir consequências lógicas a partir de um conjunto de axiomas. Para utilizar o *reasoner*, foram definidos conceitos que possuem as restrições apresentadas na Tabela 7. Por exemplo, se uma ferramenta de teste (instância de *PTTool*) se relacionar com pelo menos uma instância de *CommunicationProtocol* por meio da propriedade *generateLoadOn*, então é possível classificá-la como uma ferramenta capaz de gerar carga (*LoadGenerationTool*). Ainda, o conceito *HTTPLoadTestTool* representa as ferramentas que podem gerar carga no protocolo HTTP; o conceito *MonitoringTool* agrupa ferramentas que podem monitorar pelo menos uma instância de *MachinePerformanceMetric*; o conceito *LinuxMonitoringTool* classifica as ferramentas que monitoram pelo menos uma instância do sistema operacional Linux; entre outros conceitos ilustrados na Tabela 7.

Tabela 7: Restrições dos conceitos definidos utilizados para aplicar o *reasoner* sobre instâncias da ontologia.

Conceito	Restrições lógica do conceito
LoadGenerationTool	$PTTool \wedge \text{generatesLoadOn some } CommunicationProtocol$
HTTPLoadTestTool	$LoadGenerationTool \wedge \text{generatesLoadOn value } HTTP$
SOAPLoadTestTool	$LoadGenerationTool \wedge \text{generatesLoadOn value } SOAP$
MonitoringTool	$PTTool \wedge \text{canCollectMetric some } MachinePerformanceMetric \wedge \text{supports some } PerformanceMetricMonitoring$
LoadGenerationAndMonitoringTool	$LoadGenerationTool \wedge MonitoringTool$
LinuxMonitoringTool	$MonitoringTool \wedge \text{canMonitor some } LinuxBasedOperatingSystem$
WindowsMonitoringTool	$MonitoringTool \wedge \text{canMonitor some } WindowsBasedOperatingSystem.$
ToolAvailableForLinux	$PTTool \wedge \text{worksOn some } LinuxBasedOperatingSystem.$
ToolAvailableForWindows	$PTTool \wedge \text{worksOn some } WindowsBasedOperatingSystem.$

Estes conceitos apresentados na Tabela 7 não possuem instâncias, contudo um *reasoner* pode ser aplicado para inferir se alguma instância pode ser classificada nestes conceitos. Foi aplicado o *reasoner* Pellet [22] sobre a ontologia e o resultado da inferência pode ser visualizado na Figura 39. Por exemplo, das 18 instâncias de *PTTool*, 15 foram classificadas como *LoadGenerationTool*, 14 instâncias são do tipo *HTTPLoadTestTool*, 10 foram classificadas como *MonitoringTool* e assim por diante.

**Figura 39:** Resultados obtidos pela aplicação do *reasoner* Pellet sobre as instâncias da ontologia.

6.4. Utilização da Ontologia no WebProtégé

O WebProtégé é um projeto *open source* que utiliza o Protégé para oferecer as funcionalidades de desenvolvimento de ontologias e o Google Web Toolkit na interface de usuário [68]. Além de oferecer suporte a edição colaborativa, o WebProtégé permite que as informações da ontologia sejam facilmente acessadas pelos clientes web [60]. Uma das vantagens oferecida pelo WebProtégé é tornar uma ontologia disponível na web, de forma que os usuários possam navegar nela sem a necessidade de instalar softwares específicos para se trabalhar com ontologias [68]. Com o objetivo de testar se a ontologia proposta poderia também ser utilizada no WebProtégé, o Tomcat³ foi instalado para hospedar uma versão do WebProtégé que incluía a ontologia proposta. A Figura 40 ilustra a tela inicial da aplicação incluindo a ontologia denominada “Performance Test Ontology”.

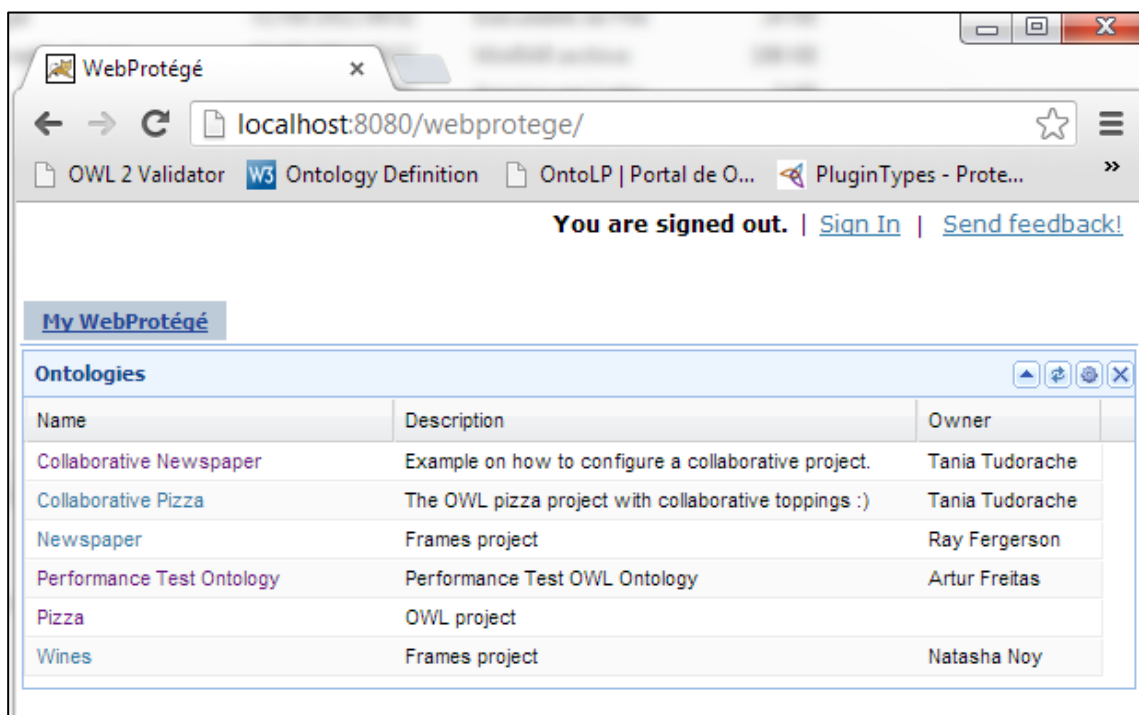


Figura 40: Tela inicial do WebProtégé hospedando a ontologia de teste de desempenho.

A configuração padrão do WebProtégé exibe 4 abas: *Classes*, *Properties*, *Individuals* e *Notes and Discussions*. Quanto a navegação na ontologia, a Figura 41 apresenta as informações disponíveis na aba *Individuals*: as classes da ontologia podem ser visualizados na parte esquerda da figura; a parte central exibe os indivíduos do conceitos selecionado (que no caso é *PTTool*); e as propriedades do indivíduo selecionado (no caso é *LoadRunner*) podem ser visualizadas na parte direita da Figura 41.

³ Disponível em <http://tomcat.apache.org/>

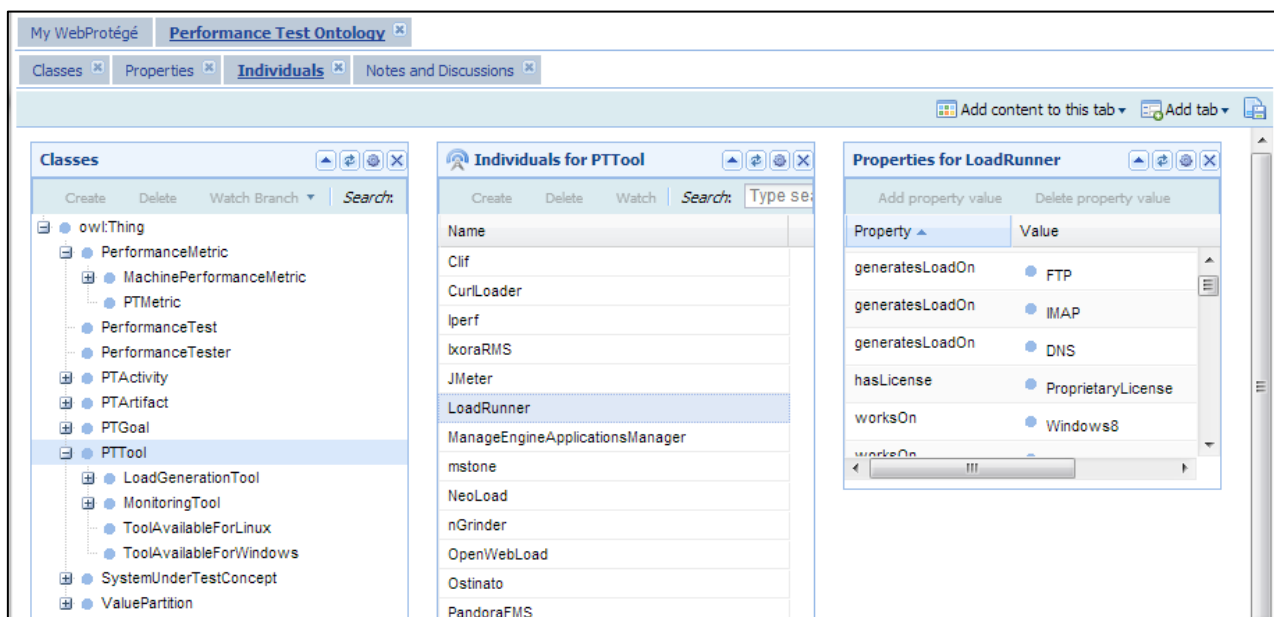


Figura 41: WebProtégé exibindo as propriedades do indivíduo *LoadRunner* (pertencente ao conceito *PTTool*).

O WebProtégé possui uma interface web customizável para navegação e edição de ontologias para facilitar a utilização por especialistas do domínio, que podem não possuir experiência com ontologias [60]. Esta afirmação é confirmada pelos menus “*Add content to this tab*” e “*Add tab*” na Figura 41. Além disso, é possível configurar os privilégios de cada usuário em relação a visualização e edição das ontologias. A Figura 42 apresenta as informações da aba *Classes*, que possui a hierarquia de conceitos na parte esquerda e as propriedades e restrições para o conceito selecionado (na imagem foi selecionado o conceito *PerformanceBottleneckIdentification*).

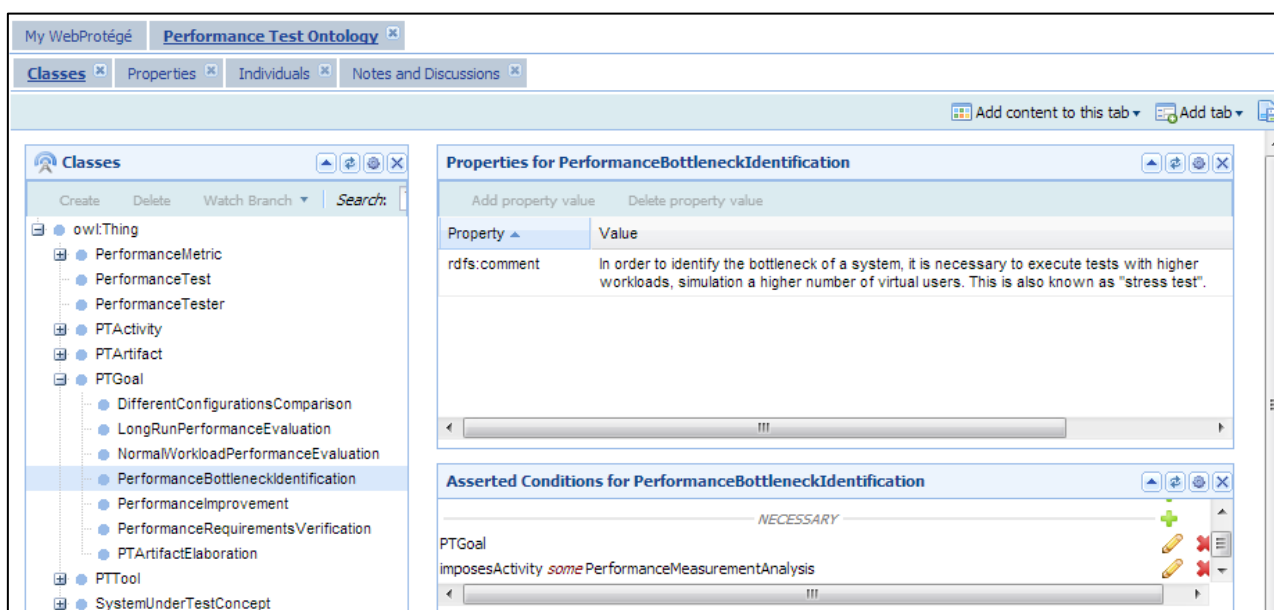


Figura 42: WebProtégé exibindo as informações do conceito *PerformanceBottleneckIdentification*.

6.5. Aplicação Desenvolvida Baseada na Ontologia

Também foi desenvolvida uma aplicação que utiliza a ontologia proposta, por meio da OWL API [45]. Como os axiomas estão codificados em um arquivo OWL gerado pelo Protégé, a aplicação consulta as informações representadas neste arquivo. Para manipular os axiomas da ontologia, a aplicação Java [55] utiliza a OWL API [45], que é uma biblioteca Java *open source* para a criação, manipulação e serialização de ontologias OWL [66]. A aplicação permite criar novas instâncias do conceito teste de desempenho, vincular uma instância específica do conceito teste de desempenho com instâncias de ferramentas de teste, atividades, objetivos, métricas, entre outros. Os conceitos e propriedades do domínio representados na ontologia podem ser criados, recuperados e atualizados na aplicação. Além disso, os axiomas relacionados a uma instância específica de teste de desempenho são utilizados pela funcionalidade de geração do plano de teste. Para criar o plano de teste no formato PDF foi utilizada a iText API [30] versão 5.2.1, uma biblioteca *open source* para manipulação de documentos PDF em Java. A interface gráfica pode ser visualizada na Figura 43, onde uma instância de *PerformanceTest* está selecionada e suas propriedades estão destacadas na interface.

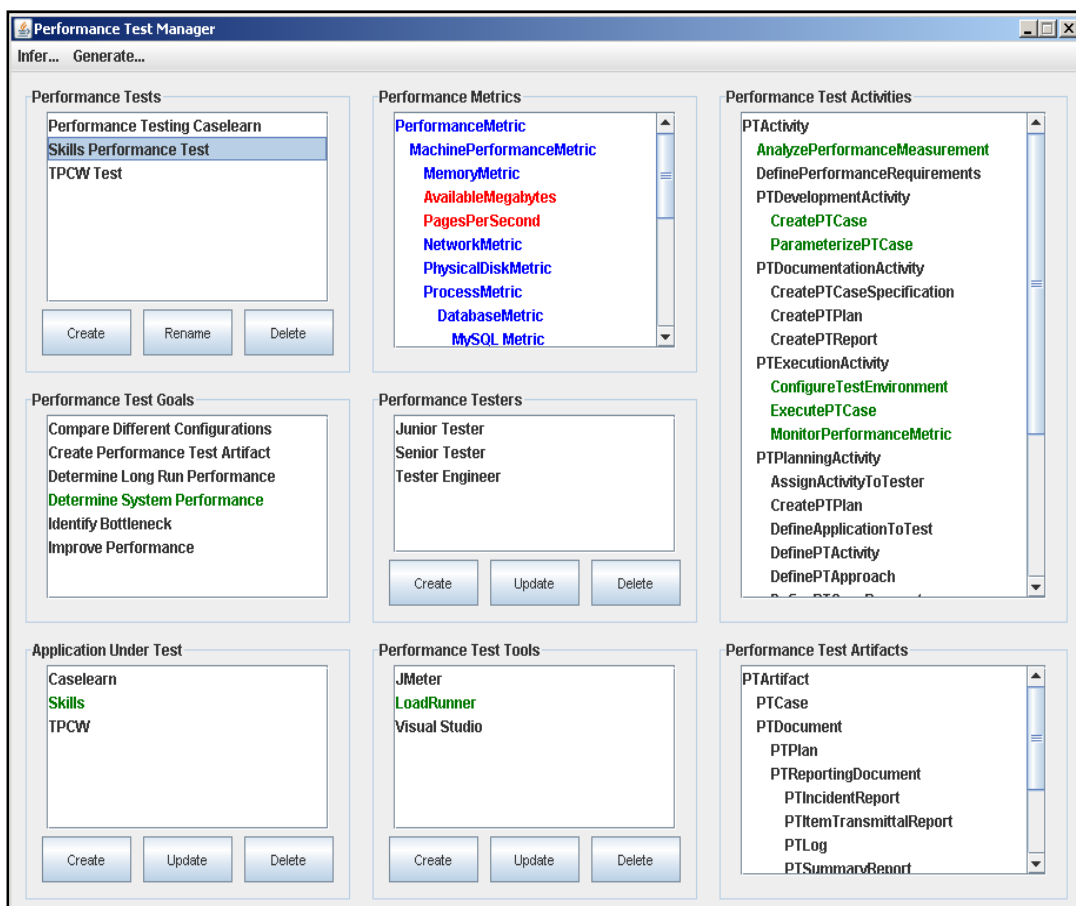


Figura 43: Interface da aplicação para planejamento e gestão de testes de desempenho.

6.5.1. Utilização da OWL API

Esta seção tem como objetivo explicar e exemplificar como os métodos da OWL API foram utilizados para carregar e manipular uma ontologia OWL. Como exemplo de uso, será demonstrado como carregar uma ontologia de um arquivo, criar um novo indivíduo e adicionar o axioma de que este indivíduo pertence a um conceito da ontologia. A aplicação apresentada utiliza este código para criar instâncias de conceitos como *PerformanceTest*, *PTActivity* e *PTTool*. A Figura 44 ilustra o código e apresenta uma linha de comentário antes de cada comando para explicar os métodos utilizados. Estes trechos de código são explicados em maior detalhe a seguir. Para acessar e manipular ontologias com a OWL API é preciso criar um objeto da classe *OWLOntologyManager*:

```
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
```

Na documentação *javadoc* da OWL API [45], a classe *OWLOntologyManager* é dita como responsável por gerenciar um conjunto de ontologias, sendo o ponto principal para criá-las e acessá-las. Este exemplo carrega a ontologia de um arquivo, contudo a OWL API também permite que a ontologia seja carregada a partir da web por meio de um IRI (Internationalized Resource Identifier). Para carregar a ontologia de um arquivo, é preciso possuir uma referência que aponte para a localização de um arquivo OWL:

```
File file = new File("File Name.owl");
```

Possuindo os dois objetos criados anteriormente, é possível carregar a ontologia. Na OWL API, a ontologia é representada pela classe *OWLOntology*, que consiste de um conjunto de axiomas representados pela classe *OWLAxiom*. O seguinte código inicializa um objeto *OWLOntology*:

```
OWLOntology ontology = manager.loadOntologyFromOntologyDocument(file);
```

Os conceitos, indivíduos, propriedades e axiomas da ontologia podem ser criados ou carregados por métodos disponibilizados pela classe *OWLDataFactory*. Um objeto do tipo *OWLDataFactory* é obtido com o seguinte código:

```
OWLDataFactory factory = manager.getOWLDataFactory();
```

Cada componente da ontologia é identificado por um IRI, que é formado pela composição do *namespace* da ontologia com a identificação do componente. O código a seguir cria um IRI para representar uma referência de um indivíduo:

```
IRI iri = IRI.create("Namespace#Indiv_Id");
```

Possuindo o IRI do indivíduo que se deseja criar, é possível criar uma referência para um objeto do tipo `OWLNamedIndividual` por meio do seguinte trecho de código:

```
OWLNamedIndividual newInstance = factory.getOWLNamedIndividual(iri);
```

Assim como cada instância deve possuir um IRI, os conceitos também são identificados por IRI. É preciso criar um IRI para o conceito que o indivíduo pertencerá:

```
IRI iri_c = IRI.create("Namespace#Concept_Id");
```

Os conceitos de uma ontologia são representados pela classe `OWLClass`, como demonstra o código abaixo:

```
OWLClass owlClass = factory.getOWLClass(iri_c);
```

O axioma que afirma que um indivíduo pertence a um determinado conceito é representado por um objeto da classe `OWLClassAssertionAxiom`:

```
OWLClassAssertionAxiom assertion  
    = factory.getOWLClassAssertionAxiom (owlClass,newInstance);
```

Para adicionar o novo axioma na ontologia e salvar as modificações, é preciso executar, os métodos `addAxiom` e `saveOntology` da classe `OWLOntologyManager`:

```
manager.addAxiom(ontology,assertion);  
manager.saveOntology(ontology);
```

```
//create a reference to the OWL ontology file  
File owlFile = new File("FileName.owl");  
//create an OWL manager to handle ontology loading and saving operations  
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();  
//create an OWL ontology reference loading from the specified file  
OWLOntology ontology = manager.loadOntologyFromOntologyDocument(owlFile);  
//retrieve an instance of the OWLDataFactory class  
OWLDataFactory factory = manager.getOWLDataFactory();  
//create an IRI (Internationalized Resource Identifier) of an individual  
IRI instanceIri = IRI.create("Namespace#Individual_Identification");  
//create an individual according to its IRI  
OWLNamedIndividual newInstance = factory.getOWLNamedIndividual(instanceIri);  
//create an IRI of a concept  
IRI classIri = IRI.create("Namespace#Class_Identification");  
//create a reference to the concept according to its IRI  
OWLClass owlClass = factory.getOWLClass(classIri);  
//create an axioms to assert that the individual belongs to the class  
OWLClassAssertionAxiom assertion =  
    factory.getOWLClassAssertionAxiom(owlClass, newInstance);  
//add the axiom in the ontology  
manager.addAxiom(ontology, assertion);  
//save the ontology, updating the changes in the OWL file  
manager.saveOntology(ontology);
```

Figura 44: Código fonte para criar uma instância de um conceito na ontologia utilizando a OWL API.

Esta seção apresentou um exemplo de como carregar uma ontologia utilizando a OWL API, como criar um novo indivíduo e adicionar o axioma de que este indivíduo pertence a um determinado conceito da ontologia. A próxima seção deste trabalho explica um exemplo de uso da API utilizada para geração dos planos de teste em PDF.

6.5.2. Exemplo de Uso da iText API

Este trabalho utilizou o iText⁴ versão 5.2.1 para geração do plano de teste em formato PDF. O iText é uma API desenvolvida em Java para criação e manipulação de documentos PDF. A Figura 45 demonstra um trecho de um plano de teste em PDF gerado contendo as informações referentes à instância do conceito teste de desempenho denominada “Skills Performance Test”.

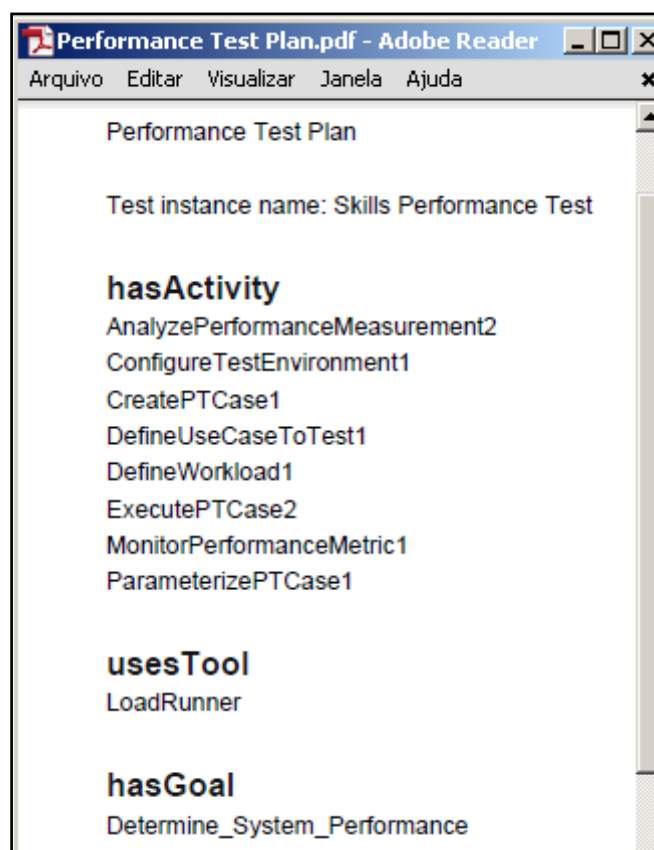


Figura 45: Plano de teste para a instância de *PerformanceTest* denominada “Skills Performance Test”.

Para gerar o plano de teste para um indivíduo do tipo teste selecionado, o algoritmo inicia com um objeto da classe *OWLIndividual* (da OWL API) que representa o teste desejado e executa os seguintes passos:

⁴ Disponível em <http://sourceforge.net/projects/itext/>

- 1) Descobrir todas as propriedades do teste por meio do método *getObjectPropertyValues* existente na classe *OWLIndividual*.
- 2) Para cada propriedade P, escrever a identificação da propriedade P no PDF e cada indivíduo da ontologia relacionado pela propriedade P com o teste em questão.

Visando demonstrar a utilização da iText API, o código ilustrado na Figura 46 cria o arquivo "Test plan.pdf" contendo o parágrafo "Hello World".

```
File file = new File ("Test plan.pdf");
FileOutputStream fos = new FileOutputStream(file);
Document document = new Document();
PdfWriter.getInstance(document, fos);
document.open();
document.add(new Paragraph("Hello World"));
document.close();
```

Figura 46: Exemplo de uso da iText para criar um PDF.

Esta seção demonstrou diferentes aplicações baseadas na ontologia proposta. Para o desenvolvimento destas aplicações, foram estudadas tecnologias como, por exemplo, Protégé [62], Pellet [22], Java [55], OWL API [66], SQWRL [47], JUNG [37] e iText [30]. A seção seguinte apresentará as considerações finais desta pesquisa, incluindo uma comparação das contribuições deste trabalho com os trabalhos relacionados.

7. CONSIDERAÇÕES FINAIS

"A ciência se compõe de erros que, por sua vez, são os passos em direção à verdade." (Júlio Verne)

Inicialmente, esta seção de considerações finais destaca as principais contribuições resultantes desta pesquisa de modo a compará-las com as contribuições dos trabalhos relacionados. Uma visão geral desta comparação é apresentada na Tabela 8. Os títulos dos 19 trabalhos que foram comparados estão listados a seguir:

- T1)** Ontologia para teste de desempenho de software (este trabalho)
- T2)** How far are we from the definition of a common software performance ontology? [69]
- T3)** Performance-related ontologies and semantic web applications for on-line performance assessment of intelligent systems [29]
- T4)** Web performance and behavior ontology [11]
- T5)** Using ontologies to improve performance in a web system – a web caching system case of study [10]
- T6)** An approach to ontology-aided performance engineering through NFR framework [56]
- T7)** Ontology-based web application testing [61]
- T8)** Towards the establishment of an ontology of software testing [19]
- T9)** Knowledge-based software test generation [71]
- T10)** Test case reuse based on ontology [39]
- T11)** Research and implementation of knowledge management methods in software testing process [41]
- T12)** Ontology-based web service robustness test generation [42]
- T13)** An ontology-based software test generation framework [70]
- T14)** Ontology-based test case generation for simulating complex production automation systems [67]
- T15)** Ontology-based test generation for multiagent systems [12]
- T16)** Developing a software testing ontology in UML for a software growth environment of web-based applications [26]
- T17)** A multi-agent software environment for testing web-based applications [57]
- T18)** Ontology-based test case generation for testing web services [74]
- T19)** SwTO¹ (Software Test Ontology Integrated) and its Application in Linux Test [13]

Os trabalhos relacionados anteriormente listados foram comparados de acordo com as seguintes contribuições:

- C1)** Apresenta uma ontologia
- C2)** Apresenta no mínimo uma aplicação baseada em ontologia
- C3)** Inclui avaliação ou validação de uma ontologia
- C4)** Compara duas ou mais ontologias
- C5)** Compara trabalhos sobre ontologias
- C6)** Cita (ou utiliza) referências para criar uma ontologia
- C7)** Cita a aplicação de tecnologias relacionadas à área de ontologia

Tabela 8: Comparação resumida das contribuições dos trabalhos relacionados.

Contribuição Trabalho	C1	C2	C3	C4	C5	C6	C7	Total	Ano da publicação
T1	✓	✓	✓	✓	✓	✓	✓	7	2013
T2	✗	✗	✗	✗	✗	✓	✗	1	2005
T3	✓	✓	✗	✗	✗	✓	✓	4	2006
T4	✓	✓	✓	✗	✗	✓	✓	5	2008
T5	✓	✓	✓	✗	✗	✓	✓	5	2008
T6	✓	✓	✗	✗	✗	✓	✓	4	2007
T7	✗	✗	✗	✗	✓	✓	✗	2	2008
T8	✓	✓	✗	✗	✗	✓	✓	4	2006
T9	✗	✓	✗	✗	✗	✗	✓	2	2009
T10	✓	✓	✗	✗	✗	✓	✓	4	2009
T11	✓	✓	✗	✗	✗	✓	✗	3	2009
T12	✗	✓	✓	✗	✗	✗	✓	3	2009
T13	✗	✓	✗	✗	✗	✓	✗	2	2010
T14	✗	✓	✗	✗	✗	✗	✓	2	2010
T15	✓	✓	✗	✗	✗	✗	✓	3	2008
T16	✓	✓	✗	✗	✗	✗	✓	3	2005
T17	✓	✓	✗	✗	✗	✗	✓	3	2003
T18	✗	✓	✗	✗	✗	✗	✓	2	2007
T19	✓	✓	✓	✗	✗	✓	✓	5	2009
Total	12	17	5	1	2	12	15		

De acordo com a Tabela 8, é possível observar que:

- A contribuição mais frequente, em 17 dos 19 trabalhos, é a proposta de uma aplicação baseada em ontologias. Contudo, apenas 12 trabalhos apresentam uma ontologia;
- A avaliação ou validação de uma ontologia não é comum de acontecer, uma vez que apenas 5 dos 19 trabalhos apresentaram uma contribuição deste tipo;
- Comparações entre duas ou mais ontologias não foram encontradas em trabalhos relacionados, entretanto nesta dissertação são comparadas 3 ontologias;
- A comparação de trabalhos relacionados às áreas de ontologia e teste de software é uma contribuição que foi identificada em apenas 2 dos 19 trabalhos analisados;
- Dos 19 trabalhos, 12 citam referências que podem ser ou foram utilizadas para criar uma ontologia. Os trabalhos T2, T7 e T13 não criaram ontologias, mas citaram estas referências. Por outro lado, os trabalhos T15, T16 e T17 criaram uma ontologia sem citar se ou quais referências foram utilizadas para isto;
- Embora 17 trabalhos apresentem aplicações de ontologias, apenas 15 comentaram utilizar tecnologias relacionadas à área de ontologia.

A Tabela 9 e a Tabela 10 aumentam o nível de detalhamento sobre a informação apresentada na Tabela 8. De acordo com a Tabela 9, podemos perceber que:

- A mesma ontologia é apresentada nos trabalhos T4 e T5, assim como os trabalhos T15 e T16 também se referem a mesma ontologia;
- As aplicações baseadas em ontologias são bem diversificadas, sendo o tema mais recorrente a geração de teste. Contudo, mesmo neste tema, o foco varia de trabalho para trabalho (teste funcional, robustez, de *web service*).
- As formas de avaliação ou validação encontradas foram por meio do cálculo de métricas da ontologia, questionando especialistas do domínio ou simulando o uso da aplicação baseada na ontologia.

A Tabela 10 permite concluir que:

- A comparação entre ontologias e entre trabalhos relacionados ainda é incipiente;
- A referência mais citada para criação de uma ontologia (5 votos) é o SWEBOK [1];
- A tecnologia mais utilizadas, por 9 dos 19 trabalhos, é OWL. Além disso, OWL-S, que é uma variação de OWL, é utilizada por mais 2 trabalhos. A segunda tecnologia relacionada a ontologia mais aplicada é o Protégé, citado por 7 trabalhos.

Tabela 9: Comparação das contribuições dos trabalhos relacionados (parte 1).

	C1	C2	C3
T1	Performance Test Ontology	Planejamento dos testes de desempenho	Métricas da ontologia são calculadas e especialistas do domínio são questionados
T2	✘	✘	✘
T3	Performance Ontology	Melhoria do desempenho durante a execução	✘
T4	Web Performance and Behavior Ontology		Uso da aplicação proposta é simulado
T5			
T6	Softgoal Interdependency Graph Ontology	Gerenciamento dos requisitos não funcionais visando otimização do desempenho	✘
T7	✘	✘	✘
T8	OntoTest (Ontology of Software Testing)	Estabelecer uma arquitetura de referência e um vocabulário sobre teste de software	✘
T9	✘	Geração de casos de teste unitário (funcional)	✘
T10	Software Testing Ontology	Reuso de casos de teste	✘
T11	Knowledge Ontology of Software Testing	Gerenciar o conhecimento de teste de software	✘
T12	✘	Geração de dados para testar a robustez de <i>web services</i>	Uso da aplicação proposta é simulado
T13	✘	Geração de casos de teste	✘
T14	✘		✘
T15	Agent Interaction Ontology	Permitir a comunicação entre agentes que possuem como objetivo gerar dados de teste	✘
T16	Software Testing Ontology in UML-XML	Permitir a comunicação entre agentes que têm objetivo de gerar e executar testes	✘
T17			✘
T18	✘	Geração de casos de teste para <i>web services</i>	✘
T19	OSOnto (Operating System Ontology), SwTO (Software Test Ontology) e SwTOI (SwTO Integrated)	Geração de sequências de teste para o Linux	Métricas da ontologia são calculadas e especialistas do domínio são questionados

Tabela 10: Comparação das contribuições dos trabalhos relacionados (parte 2).

	C4	C5	C6	C7
T1	Performance Test Ontology, SwTO e OntoTest	Compara 19 trabalhos	SWEBOK, IEEE Std. 829 e IEEE Std. 610.12	OWL, Protégé, WebProtégé, OWL API, Pellet, SQWRL
T2	✘	✘	UML SPT, Core Scenario Model e SPE-MM	✘
T3	✘	✘	UML SPT	OWL, Protégé, SWRL, Jess, Jena, Kazuki
T4	✘	✘	RFC 2616 (HTTP/1.1)	OWL, Jastor
T5	✘	✘		
T6	✘	✘	NFR Framework Softgoal Interdependency Graph	OWL
T7	✘	Compara 11 trabalhos	SWEBOK, IEEE Std. 829, ISO/IEC 15939:2007 e ISO/IEC 9126-1:2001	✘
T8	✘	✘	ISO/IEC 12207	Ontologia em UML e XML
T9	✘	✘	✘	OWL, POSL, OO jDREW
T10	✘	✘	ISO/IEC 9126-1:2001 e SWEBOK	OWL, Protégé
T11	✘	✘	SWEBOK	✘
T12	✘	✘	✘	OWL-S
T13	✘	✘	OMG Ontology Definition Meta Model	✘
T14	✘	✘	✘	Protégé
T15	✘	✘	✘	OWL, Protégé, JADE
T16	✘	✘	✘	Ontologia em UML e XML
T17	✘	✘	✘	
T18	✘	✘	✘	OWL-S, Protégé, Jena2
T19	✘	✘	SWEBOK, Basic Linux Ontology	OWL, Protégé, Jena, Racer

Este trabalho propôs uma ontologia sobre teste de desempenho de software e explorou aplicações que a utilizam. Para a construção da ontologia, foram estudadas técnicas, ferramentas e metodologias da engenharia ontológica, conforme apresentado na

fundamentação teórica. Os conceitos, relações, propriedades, axiomas e instâncias que representam o conhecimento da ontologia foram extraídos da bibliografia de testes de desempenho e de ontologias de domínios relacionados. A ontologia proposta é utilizada em aplicações projetadas para auxiliar o gerenciamento e o planejamento dos testes de desempenho. Ontologias são representações formais do conhecimento que permitem o compartilhamento do conhecimento do domínio entre diferentes aplicações. Assim, outras aplicações podem ser exploradas com base na mesma ontologia, o que resulta em interoperabilidade e um “comprometimento ontológico”. Este comprometimento é um acordo onde fica especificado que será utilizado um vocabulário consistente com os significados definidos em uma ontologia. Portanto, utilizar uma abordagem ontológica permite padronizar o vocabulário da área e pode facilitar a troca de conhecimentos e a comunicação entre testadores, gerentes, desenvolvedores e usuários [19]. Além disso, ontologias já foram pesquisadas para a geração de testes [71], melhoramento do desempenho em tempo de execução [11], para o estabelecimento de um processo de aprendizagem sobre teste de software [19], entre outras aplicações. Este estudo apresentou o uso de ontologias no planejamento e especificação dos testes de software. A motivação da pesquisa foi investigar se ontologia, como técnica de representação do conhecimento, pode melhorar o processo de teste de desempenho, auxiliando a tomada de decisão e aumentando a qualidade dos testes. A ontologia representa tanto o conhecimento sobre o domínio de teste de desempenho, quanto o conhecimento de testes anteriores. Portanto, uma abordagem ontológica pode apoiar a aquisição e o compartilhamento de conhecimentos sobre novas tecnologias, como novas ferramentas de teste. Além disso, ao longo desta pesquisa foram investigadas e aplicadas as seguintes tecnologias: Protégé [62], Pellet [22], Java [55], OWL API [66], SQWRL [47], JUNG [37] e iText [30].

O objetivo geral deste trabalho foi a construção de uma ontologia sobre teste de desempenho de software. Como consequência, também foi explorada a aplicação desta ontologia nas atividades de planejamento dos testes de desempenho. Como vantagens desta abordagem, é possível perceber que a utilização de ontologias pode proporcionar uma maior organização e gerenciamento do conhecimento nos testes de desempenho. Sobre a ontologia de teste de desempenho proposta, foram elaboradas inferências como, por exemplo, deduzir quais tecnologias podem ser utilizadas para testar determinada aplicação em um dado ambiente de teste. Em um nível mais alto de abstração, as aplicações podem auxiliar o testador a definir o que deve ser levado em consideração durante a elaboração de um teste de desempenho de software. Assim, a ontologia pode

auxiliar o gerenciamento das instâncias dos conceitos e propriedades no domínio, como ferramentas de teste de desempenho, atividades, métricas, objetivos e artefatos. As aplicações baseadas nos axiomas da ontologia podem ser utilizadas, por exemplo, para indicar atividades e ferramentas alinhadas aos objetivos do teste.

A ontologia proposta foi comparada com duas ontologias relacionadas e validada por especialistas no domínio de teste de desempenho. Segundo [49], uma ontologia não deve conter todas as informações possíveis sobre um domínio uma vez que a conceitualização não precisa exceder as necessidades das aplicações que utilizam a ontologia. Além disso, uma ontologia é um modelo abstrato da realidade e possui como limitação a capacidade de representar apenas parcialmente as informações existentes em um domínio de conhecimento. Em outras palavras, embora os principais termos do domínio de teste de desempenho tenham sido representados, existem mais conceitos, propriedades e instâncias que atualmente não fazem parte da ontologia proposta. Contudo, a qualquer momento é possível criar novos conceitos para expandir a ontologia; um exemplo disso é o conceito *PerformanceTest* que pode ser especializado pelas classes *LoadTest*, *StressTest*, *EnduranceTest*. Um *LoadTest* pode ser definido como um *PerformanceTest* que deve possuir como objetivo a avaliação do desempenho sob condições normais de carga, uma instância de *StressTest* apresentará o objetivo de identificar o gargalo do sistema e um *EnduranceTest* é um teste cujo objetivo é avaliar o desempenho por longos períodos de interação. Os objetivos citados anteriormente são conceitos que já existem na ontologia, mas, como mostrado, novos conceitos podem ser adicionados. Outro exemplo é o conceito *PerformanceTester*, que pode receber especializações segundo critérios de uma empresa que utilize a ontologia, como *TestAnalyst*, *TestDeveloper*, *TestEngineer*, *SeniorTester*, etc. Além disso, seria interessante utilizar uma técnica chamada *modular design*, que visa separar uma ontologia grande em ontologias menores (módulos) visando aumentar o reuso e facilitar a importação apenas das partes desejadas. Neste caso, a ontologia sobre o domínio de teste de desempenho poderia ser a composição de uma ontologia das atividades de teste, uma ontologia sobre o sistema sob teste, uma ontologia sobre as ferramentas de teste, e assim por diante.

A ontologia foi desenvolvida para atender os casos de uso explicados na Seção 4, contudo é possível ampliar a conceitualização da ontologia para novas funcionalidades de interesse aos testes de desempenho. A identificação de sequências de interações mais relevantes para um teste e a identificação de gargalos nas métricas de desempenho são exemplos de aplicações que podem ser exploradas como trabalhos futuros. Ainda, é

possível explorar como a ontologia poderia ser expandida para permitir a criação de um caso de teste de desempenho conceitualmente e, após, converter este caso de teste para uma determinada ferramenta. Além disto, seria interessante investigar como a ontologia poderia converter um caso de teste já implementado de uma ferramenta para outra. Também, a ontologia pode ser futuramente utilizada em aplicações visando auxiliar o testador com a geração de outros documentos além do plano de teste.

As atividades planejadas para o mestrado foram concluídas, a saber:

- estudar trabalhos relacionados e comparar suas contribuições;
- explorar o estado da arte sobre ontologias, teste de software e teste de desempenho;
- construir a ontologia sobre o domínio de teste de desempenho;
- validar o conhecimento da ontologia por meio de especialistas;
- investigar e desenvolver aplicações baseadas na ontologia;
- comparar a ontologia proposta com ontologias relacionadas.

Por fim, podemos perceber que o foco dos sistemas de informação está se movendo do “processamento de dados” em direção ao “processamento de conceitos” [31]. Seguindo esta conclusão, a unidade básica de processamento está se transformando de dados para conceitos semânticos que possuem uma interpretação e existem em um contexto com outros conceitos [31]. Este trabalho investigou aplicações, vantagens e limitações de utilizar um paradigma ontológico para o domínio de teste de desempenho. A ontologia proposta, as aplicações que a utilizam e toda documentação relacionada estão disponíveis para *download* no endereço: <http://code.google.com/p/performance-ontology/>.

REFERÊNCIAS

- [1] A. Abran, J. Moore, P. Bourque, R. Dupuis, L. Tripp. "Guide to the Software Engineering Body of Knowledge (SWEBOK)". Relatório Técnico, IEEE Computer Society, 2004, 202p.
- [2] A. Bastos, E. Rios, R. Cristalli, T. Moreira. "Base de Conhecimento em Teste de Software". Martins Fontes, 2007, 263p.
- [3] A. Bertolino. "Software Testing Research: Achievements, Challenges, Dreams". *Future of Software Engineering*, 2007, pp. 85-103.
- [4] A. Crespo, O. Silva, C. Borges, C. Salviano, M. Teive, A. Junior, M. Jino. "Uma Metodologia para Teste de Software no Contexto da Melhoria de Processo". *Simpósio Brasileiro de Qualidade de Software*, 2004, pp. 271-285.
- [5] A. Duineveld, R. Stoter, M. Weiden, B. Kenepa, V. Benjamins. "WonderTools? A comparative study of ontological engineering tools". *International Journal of Human-Computer Studies*, vol. 52, n. 6, 2000, pp. 1111-1133.
- [6] A. Farquhar, R. Fikes, J. Rice. "The Ontolingua Server: A Tool for Collaborative Ontology Construction". *International Journal of Human-Computer Studies*, vol. 46, n. 6, 1997, pp. 707-727.
- [7] A. Gangemi, C. Catenacci, M. Ciaramita, J. Lehmann. "A theoretical framework for ontology evaluation and validation". *Semantic Web Applications and Perspectives, Proceedings of the 2nd Italian Semantic Web Workshop*, 2005, 16p.
- [8] A. Gómez-Pérez, M. Fernández-López, O. Corcho. "Ontological engineering: with examples from the areas of knowledge management, e-commerce and the semantic web". Springer, 2004, 420p.
- [9] B. Sabata, S. Chatterjee, M. Davis, J. Sydir, T. Lawrence. "Taxonomy of QoS Specifications". *International Workshop on Object-Oriented Real-Time Dependable Systems*, 1997, pp. 100-107.
- [10] C. Guerrero, C. Juiz, R. Puigjaner. "Using ontologies to improve performance in a web system – A web caching system case of study". *International Conference on Web Information Systems and Technologies*, 2008, pp. 117-122.
- [11] C. Guerrero, C. Juiz, R. Puigjaner. "Web Performance and Behavior Ontology". *International Conference on Complex, Intelligent and Software Intensive Systems*, 2008, pp. 219-225.
- [12] C. Nguyen, A. Perini, P. Tonella. "Ontology-based Test Generation for MultiAgent Systems". *International Conference on Autonomous Agents and Multiagent Systems*, 2008, pp. 1315-1320.
- [13] D. Bezerra, A. Costa, K. Okada. "SwTOI (Software Test Ontology Integrated) and its Application in Linux Test". *International Workshop on Ontology, Conceptualization and Epistemology for Information Systems, Software Engineering and Service Science*, 2009, pp. 25-36.
- [14] D. Fensel, F. Harmelen, I. Horrocks, D. McGuinness, P. Patel-Schneider. "OIL: An Ontology

- Infrastructure for the Semantic Web". *IEEE Intelligent Systems*, vol. 16, 2001, pp. 38-45.
- [15] D. Fensel. "Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce". Springer, 2003, 172p.
- [16] D. Gasevic, D. Djuric, V. Devedzic. "Model Driven Architecture and Ontology Development". Springer-Verlag, 2006, 312p.
- [17] D. McGuinness, F. Harmelen. "OWL Web Ontology Language Overview". World Wide Web Consortium. Disponível em <http://www.w3.org/TR/2004/REC-owl-features-20040210/>. Acessado em outubro de 2011.
- [18] E. Barbosa, E. Nakagawa, A. Riekstin, J. Maldonado. "Ontology-based Development of Testing Related Tools". *International Conference on Software Engineering & Knowledge Engineering*, 2008, pp. 697-702.
- [19] E. Barbosa, E. Nakagawa, J. Maldonado. "Towards the Establishment of an Ontology of Software Testing". *International Conference on Software Engineering and Knowledge Engineering*, 2006, pp. 522-525.
- [20] E. Motta. "An Overview of the OCML Modelling Language". *Workshop on Knowledge Engineering Methods & Languages*, 1998, pp. 21-22.
- [21] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, Y. Katz. "Pellet: A Practical OWL-DL Reasoner". Relatório Técnico, University of Maryland, Institute for Advanced Computer Studies. Disponível em <http://www.mindswap.org/papers/PelletDemo.pdf>, 2005, 26p.
- [22] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, Y. Katz. "Pellet: A Practical OWL-DL Reasoner". *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, n. 2, 2007, pp. 51-53.
- [23] F. García-Peñalvo, J. García, R. Therón. "Analysis of the OWL ontologies: A survey". *Scientific Research and Essays*, vol. 6, n. 20, 2011, pp. 4318-4329.
- [24] G. Everett, R. McLeod Jr. "Software Testing: Testing Across the Entire Software Development Life Cycle". John Wiley & Sons, 2007, 345p.
- [25] G. Myers. "The Art of Software Testing". John Wiley & Sons, 2004, 256p.
- [26] H. Zhu, Q. Huo. "Developing a Software Testing Ontology in UML for A Software Growth Environment of Web-Based Applications". Cap. 9, livro *Software Evolution with UML and XML*, H. Yang (editor), Idea Group, 2005.
- [27] IEEE Computer Society. "IEEE Standard for Software Test Documentation, Std. 829". 1998, 59p.
- [28] IEEE Computer Society. "IEEE Standard Glossary of Software Engineering Terminology, Std. 610.12". 1990, 83p.
- [29] I. Lera, C. Juiz, R. Puigjaner. "Performance-related ontologies and semantic web applications for on-line performance assessment of intelligent systems". *Science of Computer Programming -*

Special issue on quality system and software architectures, vol. 61, n. 1, 2006, pp. 27-37.

- [30] “iText API”. Disponível em <http://itextpdf.com/>. Acessado em abril de 2012.
- [31] J. Brank, M. Grobelnik, D. Mladenić. “A Survey of Ontology Evaluation Techniques”. *Conference on Data Mining and Data Warehouses*, 2005, pp. 166-169.
- [32] J. Brank, M. Grobelnik, D. Mladenic. “Automatic Evaluation of Ontologies”. Cap. 11, livro *Natural Language Processing and Text Mining*, A. Kao, S. Potet (editores), Springer, 2007, pp. 193-219.
- [33] J. Domingue. “Tadzebao and WebOnto: discussing, browsing, and editing ontologies on the web”. *Proceedings of the Workshop on Knowledge Acquisition, Modeling and Management*, 1998.
- [34] J. Evermann, J. Fang. “Evaluating ontologies: Towards a cognitive measure of quality”. *Information Systems*, vol. 35, n. 4, 2010, pp. 391-403.
- [35] J. Gennari, M. Musen, R. Fergerson, W. Grosso, M. Crubézy, H. Eriksson, N. Noy, S. Tu. “The Evolution of Protégé: An Environment for Knowledge-Based Systems Development”. *International Journal of Human-Computer Studies*, vol. 58, n. 1, 2003, pp. 89-123.
- [36] J. Meier, C. Farre, P. Bansode, S. Barber, D. Rea. "Performance Testing Guidance for Web Applications: Patterns & Practices". Microsoft Press, 2007, 221p.
- [37] J. O'Madadhain, D. Fisher, P. Smyth, S. White, Y.-B. Boey. “Analysis and visualization of network data using JUNG”. *Journal of Statistical Software*, vol. 10, n. 2, 2005, pp. 1-35.
- [38] J. Uebersax. “Likert scales: dispelling the confusion”. Disponível em <http://john-uebersax.com/stat/likert.htm>. Acessado em novembro de 2012.
- [39] L. Cai, W. Tong, Z. Liu, J. Zhang, “Test Case Reuse Based on Ontology”. *IEEE Pacific Rim International Symposium on Dependable Computing*, 2009, pp. 103-108.
- [40] L. Chung, B. Nixon, E. Yu, J. Mylopoulos. "Non-functional requirements in software engineering". Springer, 1999, 472p.
- [41] L. Xue-Mei, G. Guochang, L. Yong-Po, W. Ji. “Research and Implementation of Knowledge Management Methods in Software Testing Process”. *World Congress on Computer Science and Information Engineering*, 2009, pp. 739-743.
- [42] L. Xu, Q. Yuan, J. Wu, C. Liu. “Ontology-based Web Service Robustness Test Generation”. *International Symposium on Web Systems Evolution*, 2009, pp. 59-68.
- [43] M. Cristani, R. Cuel. “A Comprehensive Guideline for Building a Domain Ontology from Scratch”. *International Conference on Knowledge Management*, 2004, pp. 205-212.
- [44] M. Fernández, A. Gómez-Peréz, J. Sierra, A. Sierra. “Building a chemical ontology using Methontology and the Ontology Design Environment”. *IEEE Intelligent Systems*, vol. 14, 1999, pp. 37-46.

- [45] M. Horridge, S. Bechhofer. “The OWL API: A Java API for OWL Ontologies”. *Semantic Web*, vol. 2, 2011, pp. 11-21.
- [46] M. Kifer, G. Lausen, J. Wu. “Logical foundations of object-oriented and frame-based languages”. *Journal of the ACM*, vol. 42, n. 4, 1995, pp. 741-843.
- [47] M. O'Connor, A. Das. “SQWRL: A Query Language for OWL”. *International Workshop on OWL: Experiences and Directions*, 2009, 8p.
- [48] M. Woodside, G. Franks, D. Petriu. “The Future of Software Performance Engineering”. *Future of Software Engineering*, 2007, pp. 171-187.
- [49] N. Noy, D. McGuinness. “Ontology Development 101: A Guide to Creating Your First Ontology”. Relatório Técnico, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, 2001, 25p.
- [50] N. Noy, R. Ferguson, M. Musen. “The knowledge model of Protégé-2000: Combining interoperability and flexibility”. *European Workshop on Knowledge Acquisition, Modeling and Management*, 2000, pp. 17-32.
- [51] N. Noy, R. Guha, M. Musen. “User Ratings of Ontologies: Who Will Rate the Raters?”. *Symposium on Knowledge Collection from Volunteer Contributors*, 2005, pp. 56-63.
- [52] Ó. Corcho, A. Gómez-Pérez. “A Roadmap to Ontology Specification Languages”. *International Conference on Knowledge Acquisition, Modeling and Management*, 2000, pp. 80-96.
- [53] Ó. Corcho, M. Fernández-López, A. Gómez-Pérez. “Methodologies, tools and languages for building ontologies. Where is their meeting point?”. *Data & Knowledge Engineering*, vol. 46, n. 1, 2003, pp. 41-64.
- [54] Ó. Corcho, M. Fernández-López, A. Gómez-Pérez. “Ontological Engineering: Principles, Methods, Tools and Languages”. Cap. 1, livro *Ontologies for Software Engineering and Software Technology*, C. Calero, F. Ruiz, M. Piattini (editores), Springer-Verlag, 2006.
- [55] Oracle. “Java Website”. Disponível em <http://www.java.com>. Acessado em março de 2012.
- [56] P. Sancho, C. Juiz, R. Puigjaner, L. Chung, N. Subramanian. “An approach to ontology-aided performance engineering through NFR framework”. *International Workshop on Software and Performance*, 2007, pp. 125-128.
- [57] Q. Huo, H. Zhu, S. Greenwood. “A Multi-Agent Software Environment for Testing Web-based Applications”. *International Computer Software and Applications Conference*, 2003, pp. 210-215.
- [58] R. MacGregor. “Inside the LOOM description classifier”. *SIGART Bulletin - Special issue on implemented knowledge representation and reasoning systems*, vol. 2, n. 3, 1991, pp. 88-92.
- [59] S. Bechhofer, F. v. Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, L. A. Stein. “OWL Web Ontology Language Reference”. Disponível em

<http://www.w3.org/TR/2004/REC-owl-ref-20040210/>. Acessado em agosto de 2011.

- [60] S. Falconer, T. Tudorache, C. Nyulas, N. Noy, M. Musen. “WebProtégé: Supporting the Creation of ICD-11”. *International Semantic Web Conference*, 2010, 4p.
- [61] S. Paydar, M. Kahani. “Ontology-Based Web Application Testing”. *International Conference on Telecommunications and Networking*, 2008, pp. 23-27.
- [62] Stanford Center for Biomedical Informatics Research. “Protégé Website”. Disponível em <http://protege.stanford.edu>. Acessado em novembro de 2011.
- [63] S. Tartir, I. Arpinar, A. Sheth. “Ontological Evaluation and Validation”. *Theory and Applications of Ontology: Computer Applications*, Springer, 2010, pp. 115-130.
- [64] S. Tartir, I. Arpinar, M. Moore, A. Sheth, B. Aleman-Meza. “OntoQA: Metric-Based Ontology Quality Analysis”. *IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources*, 2005, pp. 45-53.
- [65] T. Gruber. “Towards principles for the design of ontologies used for knowledge sharing”. *International Journal of Human-Computer Studies*, vol. 43, n. 5-6, 1995, pp. 907-928.
- [66] “The OWL API”. Disponível em <http://owlapi.sourceforge.net/>. Acessado em março de 2012.
- [67] T. Moser, G. Dürr, S. Biffel. “Ontology-Based Test Case Generation For Simulating Complex”. *International Conference on Software Engineering and Knowledge Engineering*, 2010, pp. 478-482.
- [68] T. Tudorache, J. Vendetti, N. Noy. “Web-Protégé: A Lightweight OWL Ontology Editor for the Web”. *International Workshop on OWL: Experiences and Direction*, 2008, 4p.
- [69] V. Cortellessa. “How far are we from the definition of a common software performance ontology?”. *International Workshop on Software and Performance*, 2005, pp. 195-204.
- [70] V. Nasser, W. Du, D. MacIsaac. “An Ontology-based Software Test Generation Framework”. *International Conference on Software Engineering and Knowledge Engineering*, 2010, pp. 192-197.
- [71] V. Nasser, W. Du, D. MacIsaac. “Knowledge-based Software Test Generation”. *International Conference on Software Engineering and Knowledge Engineering*, 2009, pp. 312-317.
- [72] X. Su, L. Ilebrekke. “A Comparative Study of Ontology Languages and Tools”. *International Conference on Advanced Information Systems Engineering*, 2002, pp. 761-765.
- [73] W. Borst. “Construction of engineering ontologies for knowledge sharing and reuse”. Tese de Doutorado, Dutch research school for Information and Knowledge Systems, University of Twente, 1997, 227p.
- [74] Y. Wang, X. Bai, J. Li, R. Huang. “Ontology-Based Test Case Generation for Testing Web Services”. *International Symposium on Autonomous Decentralized Systems*, 2007, pp. 43-50.

APÊNDICE A – QUESTIONÁRIO SOBRE TESTE DE DESEMPENHO

Este apêndice tem como finalidade apresentar imagens do questionário aplicado nos especialistas em teste de desempenho. O questionário foi hospedado em um site e disponibilizado de forma on-line para que cada participante o respondesse em seu próprio computador. O questionário foi dividido em duas partes, sendo a primeira parte⁵ composta de perguntas abertas sobre a área. A segunda parte⁶ do questionário contém perguntas que apresentaram os principais conceitos da ontologia e solicitaram que os participantes selecionassem a opção que melhor indicasse o quanto cada um deles está de acordo com os elementos da ontologia apresentados. A página inicial desta primeira parte do questionário, que são as instruções gerais, pode ser visualizada na Figura 47.

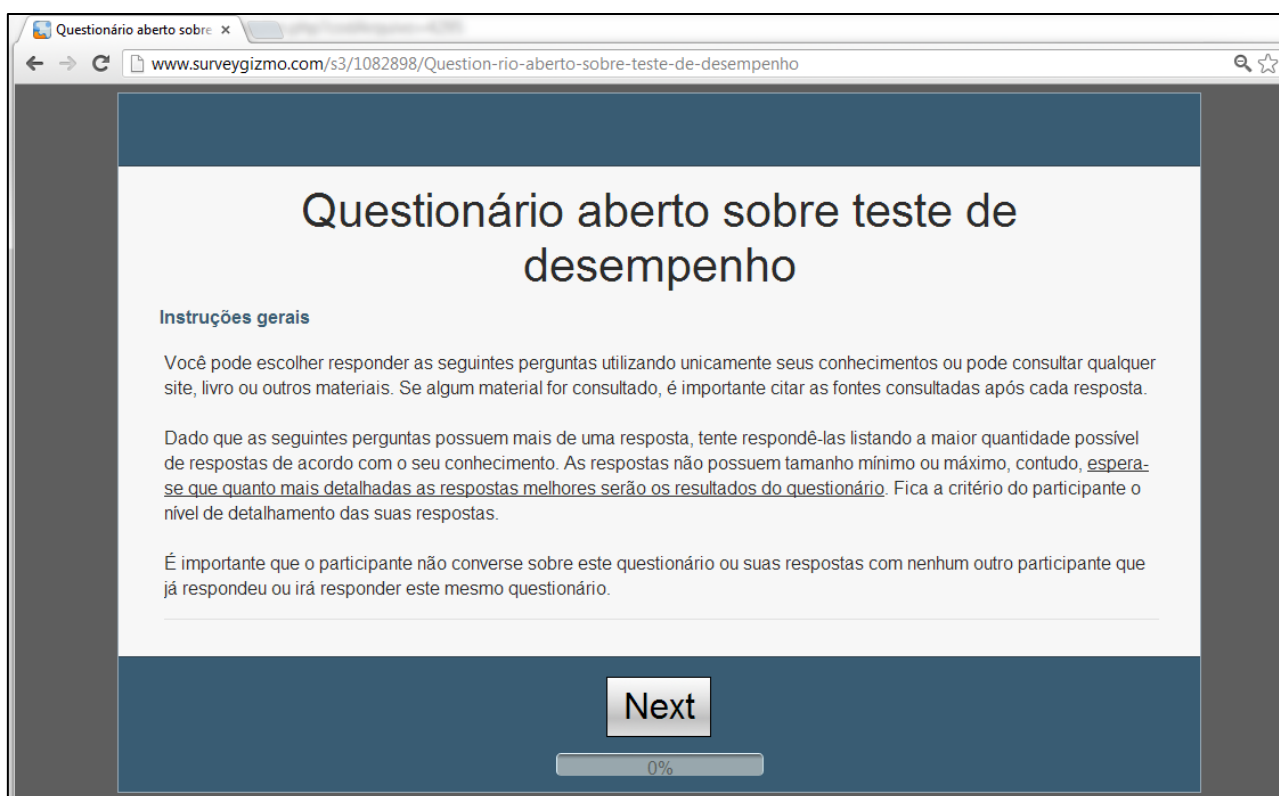


Figura 47: Instruções gerais para responder o questionário.

Após ler as instruções gerais, os participantes foram perguntados sobre qual a sua formação acadêmica, suas experiências e seus conhecimentos relacionados à área de teste de desempenho. Estas perguntas podem ser visualizadas na Figura 48.

⁵ Parte um hospedada em <http://www.surveygizmo.com/s3/1082898/Question-rio-aberto-sobre-teste-de-desempenho>

⁶ Parte dois hospedada em <http://www.surveygizmo.com/s3/1082992/Question-rio-fechado-sobre-teste-de-desempenho>

1. Qual é o seu nome? *

2. Qual é a sua formação acadêmica? (exemplo: graduação, mestrado ou doutorado em andamento ou concluído em "nome do curso") *

3. Aproximadamente há quanto tempo você estuda ou trabalha na área de teste de desempenho? *

4. Quais cargos profissionais (ou acadêmicos) relacionados com teste de desempenho você já ocupou (ou ainda ocupa) e quanto tempo aproximadamente você esteve envolvido em cada um destes cargos? *

5. Onde e como você considera que adquiriu a maior parte da sua experiência e conhecimentos sobre teste de desempenho? Você classificaria seus conhecimentos nesta área mais como práticos ou como teóricos? *

Back Next

13%

Figura 48: Informações sobre a experiência e conhecimentos do participante em teste de desempenho.

Após registrar informações sobre o nível de conhecimento e o tempo envolvido em teste de desempenho, iniciaram-se as perguntas sobre o domínio propriamente dito. A primeira pergunta foi sobre os objetivos que um teste de desempenho pode possuir, como apresentado na Figura 49. O segundo campo de resposta exibido na Figura 49 é de preenchimento opcional, apenas se o participante optar por utilizar consulta. Conforme descrito nas instruções gerais, o uso de consultas na bibliografia e na internet é opcional.

6. Quais objetivos um teste de desempenho pode possuir? *

7. Quais fontes foram consultadas para responder a pergunta anterior? (A consulta é opcional, se você preferir não consultar nenhum material poderá deixar esta questão em branco. Mas, se algum material for consultado é importante que ele seja citado aqui.)

Back Next

25%

Figura 49: Perguntas sobre os objetivos dos testes de desempenho respondidas pelos especialistas.

Semelhante a Figura 49, as demais perguntas do questionário foram as seguintes:

- Quais ferramentas existem para teste de desempenho e o que você sabe sobre elas?
- Quais atividades podem fazer parte de um teste de desempenho?
- As atividades dos testes de desempenho respondidas na questão anterior podem ser agrupadas em diferentes conjuntos? Se sim, quais?
- Quais artefatos (ou entregáveis) podem resultar de um teste de desempenho?
- Quais métricas de desempenho podem ser monitoradas?
- As métricas de desempenho podem ser agrupadas em diferentes conjuntos? Se sim, quais?

É importante lembrar que cada uma destas perguntas listadas anteriormente vinha acompanhada de um campo designado para que o especialista citasse as referências ou consultas utilizadas. Contudo, a utilização de consulta foi definida como opcional e ficava a critério de cada participante sua utilização.

Após responder as questões abertas sobre, por exemplo, as atividades de teste, as ferramentas de teste e os objetivos dos testes, os participantes foram apresentados aos conceitos existentes na ontologia que representam estes mesmos elementos. Então, os participantes foram solicitados a selecionar em uma escala de 1 a 5 (que vai de “discordo totalmente” até “concordo totalmente”) o quanto eles estão de acordo com a afirmação de que o conceito em questão faz parte do domínio de teste de desempenho. Por exemplo, na Figura 50 são apresentadas as 7 subclasses do conceito *PTGoal* existentes na ontologia e os participantes devem selecionar o quanto eles concordam com cada um dos objetivos dos testes de desempenho. Além disso, o questionário também pergunta se o participante gostaria de adicionar um novo objetivo ou remover alguns dos objetivos apresentados, como ilustra a Figura 50.

2. Você está de acordo que um teste de desempenho possa possuir os seguintes objetivos? *

	Discordo totalmente	Discordo	Não discordo e nem concordo	Concordo	Concordo totalmente
Avaliação do desempenho sob condições normais de carga *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Avaliação do desempenho durante longos períodos de interação *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Comparação do desempenho em diferentes configurações *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Elaboração de um entregável de teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Identificação de gargalos de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Melhoria do desempenho do sistema *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Verificação dos requisitos de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

3. Você adicionaria algum objetivo não citado ou removeria alguns dos objetivos acima? Se sim, quais?

Figura 50: Nível de concordância dos especialistas sobre as subclasses do conceito *PTGoal*.

Logo após, como ilustra a Figura 51, as subclasses do conceito *PTArtifact* foram apresentadas aos especialistas. Da mesma forma que na Figura 50, os participantes devem escolher, na escala Likert [38], a opção que eles julgarem mais adequada. Neste caso, a pergunta é se os conceitos apresentados existem ou não no domínio de teste de desempenho. Por último, na Figura 51 é perguntado se falta algum conceito do tipo *PTArtifact* ou ainda se um destes conceitos estaria errado.

4. Você está de acordo que um teste de desempenho possa resultar nos seguintes entregáveis (artefatos)? *

	Discordo totalmente	Discordo	Não discordo e nem concordo	Concordo	Concordo totalmente
Caso de teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Plano de teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Medições de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Relatório do teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Relatório de incidente ocorrido em um teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Registro (Log) do teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Especificação do caso de teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Especificação do procedimento de teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Especificação do projeto de teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

5. Você adicionaria algum entregável não citado ou removeria alguns dos entregáveis acima? Se sim, quais?

Figura 51: Nível de concordância dos especialistas sobre as subclasses do conceito *PTArtifact*.

De forma semelhante, na Figura 52 e na Figura 53 são apresentadas subclasses que especializam o conceito *PTActivity*. Os participantes sempre possuem um campo para justificar suas respostas, onde é perguntado se eles gostariam de adicionar novos conceitos ou remover alguns dos conceitos apresentados.

6. Você está de acordo que um teste de desempenho possa possuir as seguintes atividades? *

	Discordo totalmente	Discordo	Não discordo e nem concordo	Concordo	Concordo totalmente
Análise das medições de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Definição dos requisitos de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Desenvolvimento dos casos de teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Parametrização dos casos de teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Elaboração do plano do teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Elaboração do relatório do teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Configuração do ambiente de teste *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Execução dos casos de teste de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Monitoramento das métricas de desempenho *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Definição das atividades do teste *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Definição da abordagem do teste *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Definição dos parâmetros do caso de teste *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Definição dos objetivos do teste *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Definição do ambiente de teste *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Definição dos casos de uso do teste *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Definição da carga *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

7. Você adicionaria alguma atividade não citada ou removeria algumas das atividades acima? Se sim, quais?

Figura 52: Nível de concordância dos especialistas sobre as subclasses do conceito *PTActivity*.

8. Você está de acordo em agrupar as atividades dos testes de desempenho nas seguintes categorias? *

	Discordo totalmente	Discordo	Não discordo e nem concordo	Concordo	Concordo totalmente
Atividades de planejamento *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Atividades de desenvolvimento *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Atividades de documentação *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Atividades de execução *	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

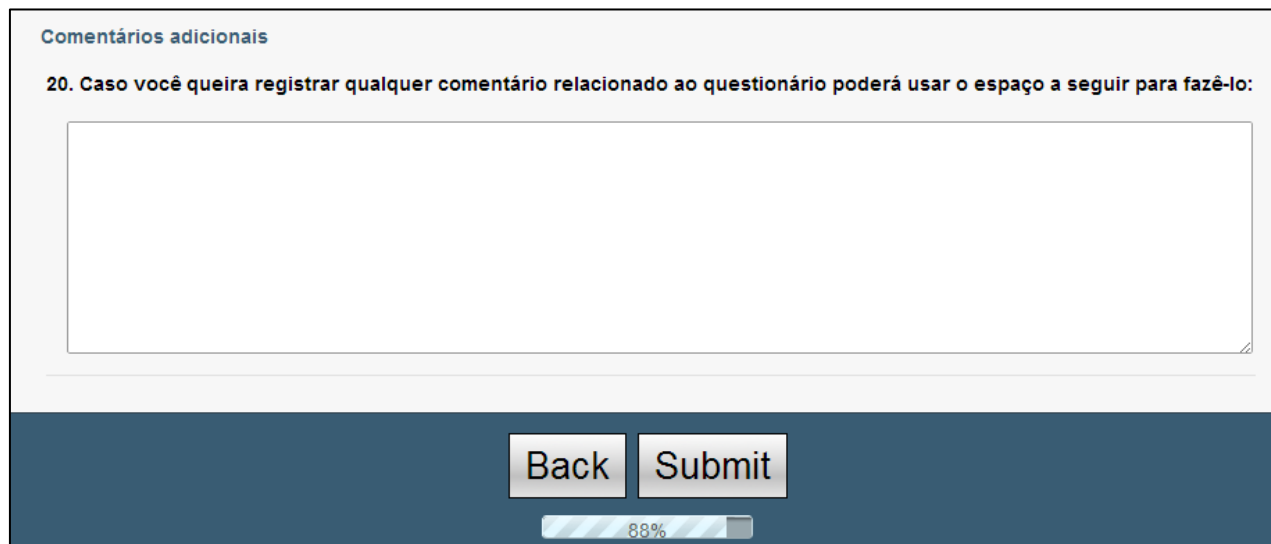
9. Você adicionaria alguma categoria não citada ou removeria algumas das categorias acima? Se sim, quais?

Back
Next

67%

Figura 53: Nível de concordância dos especialistas em relação às categorias do tipo *PTActivity*.

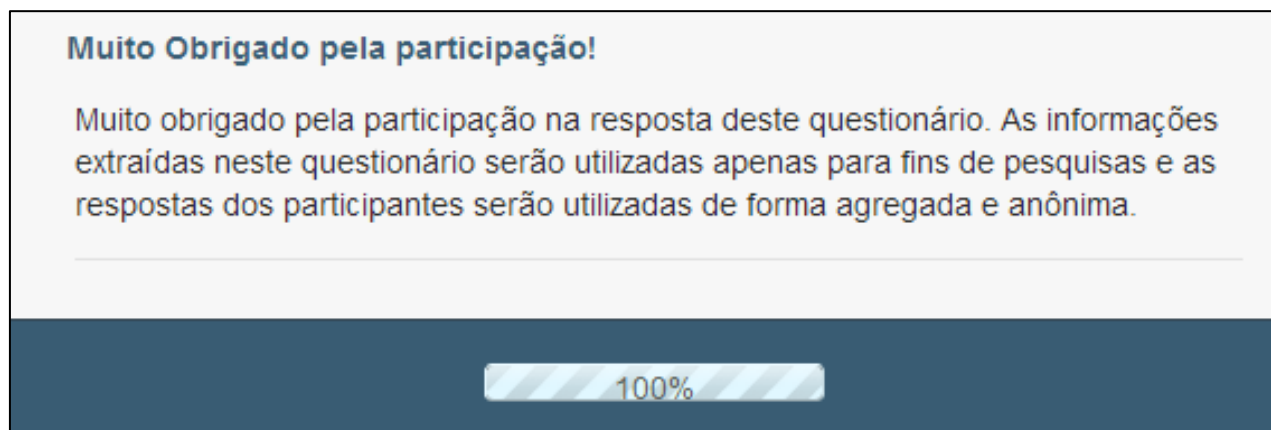
No final do questionário, um espaço foi reservado para que os especialistas que desejassem registrar comentários adicionais pudessem fazê-lo, como pode ilustrar a Figura 54.



The screenshot shows a survey interface. At the top, it says "Comentários adicionais" in blue. Below that, a question reads: "20. Caso você queira registrar qualquer comentário relacionado ao questionário poderá usar o espaço a seguir para fazê-lo:". There is a large, empty rectangular text input area. At the bottom, there are two buttons: "Back" and "Submit". Below the buttons is a progress bar showing 88% completion.

Figura 54: Opção para os especialistas registrarem comentários adicionais sobre o questionário.

Os participantes que concluíram 100% do questionário receberam a mensagem de agradecimento ilustrada na Figura 55.



The screenshot shows a thank-you message. It starts with the heading "Muito Obrigado pela participação!" in blue. The main text reads: "Muito obrigado pela participação na resposta deste questionário. As informações extraídas neste questionário serão utilizadas apenas para fins de pesquisas e as respostas dos participantes serão utilizadas de forma agregada e anônima." Below the text is a horizontal line. At the bottom, there is a progress bar showing 100% completion.

Figura 55: Mensagem de agradecimento exibida para os participantes que concluíram o questionário.

APÊNDICE B – RESPOSTAS DOS ESPECIALISTAS

Este apêndice apresenta as respostas dos especialistas em teste de desempenho referente ao questionário detalhado no Apêndice A. As respostas referentes às ocupações dos participantes em teste de desempenho e sobre a aquisição de seus conhecimentos no domínio são encontradas, respectivamente, na Tabela 11 e na Tabela 12.

Tabela 11: Ocupações dos especialistas relacionadas ao teste de desempenho.

Identificação do especialista	Ocupações relacionadas ao teste de desempenho
1	Tema da pesquisa do mestrado.
2	Tema da pesquisa do mestrado e do doutorado.
3	Estagiário no Centro de Pesquisa em Engenharia de Software (CePES -PUCRS). Atividade: desenvolver ferramentas para teste de desempenho.
4	Estagiário no CePES-PUCRS.
5	Analista de TI e doutorando em Ciência da Computação (tema de pesquisa: teste de software).
6	Estagiário no CePES-PUCRS. Atividade: suporte ao desenvolvimento de pesquisas relacionadas à área de teste de desempenho.
7	Bolsista de mestrado em um projeto de pesquisa sobre teste de desempenho.
8	Acadêmico: pesquisador/bolsista em modelagem dos testes de desempenho, geração de casos de teste/scripts de teste de desempenho e uso de tecnologias de teste de desempenho (LoadRunner e Visual Studio).

Tabela 12: Como o conhecimento em teste de desempenho foi adquirido pelos especialistas.

Identificação do especialista	Aquisição dos conhecimentos sobre teste de desempenho
1	Adquiri o conhecimento no mestrado. Como estou no primeiro ano meu conhecimento é mais teórico.
2	Como participante de um projeto de pesquisa, como aluno de mestrado e de doutorado. Meu conhecimento prático foi adquirido com as ferramentas LoadRunner, JMeter, Visual Studio e o conhecimento teórico foi por meio da leitura de artigos, relatórios técnicos e outros documentos sobre teste de desempenho. Classifico meu conhecimento prático e teórico como semelhantes.
3	Toda a experiência foi adquirida em trabalho no CePES e meus conhecimentos são na maioria práticos, visto que não tenho estudo ou formação na área.
4	Conhecimento adquiridos no CePES e experiências adquiridas na parte prática.
5	Conhecimentos práticos e teóricos na mesma proporção.
6	Adquiri maior parte dos conhecimentos no trabalho e internet. Possuo conhecimentos práticos e teóricos em mesma proporção.
7	Conhecimentos mais teóricos e adquiridos no projeto de pesquisa e no mestrado.
8	Adquiri meu conhecimento nessa área durante a realização do mestrado, engajado no projeto de pesquisa vinculada a uma empresa de TI com aplicação de teste de desempenho nos projetos de software da empresa.

As respostas sobre quais objetivos um teste de desempenho pode possuir são encontradas na Tabela 13. Apenas o especialista de número 1 utilizou consulta, que foi o livro “Performance Testing Guidance for Web Applications: Patterns & Practices”. Inclusive, este livro está nas referências desta dissertação [36] e também foi utilizado durante a definição dos conceitos e propriedades da ontologia.

Tabela 13: Objetivos dos testes de desempenho levantados pelos especialistas.

Identificação do especialista	Quais objetivos um teste de desempenho pode possuir?
1	Definir as características de desempenho do sistema, encontrar problemas de desempenho (como por exemplo, gargalos), permitir a otimização do sistema, estimar uma configuração de hardware adequada para cada aplicação e determinar a capacidade de resposta, a escalabilidade do sistema sob uma determinada carga.
2	Verificar se determinada aplicação é capaz de atender requisitos de desempenho especificados, como por exemplo: tempo de resposta da aplicação, percentual de utilização de CPU, quantidade de memória utilizada, quantidade de dados trafegados na rede e percentual de utilização do disco.
3	O teste de desempenho busca garantir a melhor performance possível de um determinado software. Isto pode ser feito por busca e remoção de erros de performance ou por prevenção de erros de performance, que são erros que causam não cumprimento de requisitos não funcionais.
4	Qualidade e confiabilidade de um determinado software. Podemos utilizar o teste de desempenho entre duas ferramentas para verificar qual delas tem maior utilidade/desempenho para funções específicas.
5	Verificar/conhecer o desempenho de uma aplicação sobre determinada carga; Definir indicadores de desempenho (ponto de partida para a definição de um SLA) Definir os recursos computacionais (hardware) necessários para que a aplicação alcance os requisitos de desempenho definidos.
6	Validar se um sistema (ou software) possui o desempenho desejado.
7	Teste de estresse ou teste de carga.
8	Avaliar se o programa satisfaz os objetivos de desempenho especificados, ou seja, se os requisitos não funcionais especificados foram atendidos com o intuito de melhorar a qualidade do produto de software. Como exemplos de indicadores para medir esse objetivo temos tempo de resposta das requisições, taxas de transferências, consumo de recursos computacionais (CPU, memória, disco, etc.), etc.

As respostas da pergunta “Quais ferramentas existem para teste de desempenho e o que você sabe sobre elas?” estão apresentadas na Tabela 14. Para responder esta questão as seguintes referências foram utilizadas:

- **Participante 1:** T. Arnold, D. Hopton, A. Leonard, e M. Frost. “Professional Software Testing with Visual Studio 2005 Team System: Tools for Software Developers and Test Engineer”. John Wiley & Sons, 2007.
- **Participante 4:** F. Campos, “Teste de Desempenho/Carga/Stress”. Disponível em <http://qualidadebr.wordpress.com/2010/08/29/teste-de-desempenhocargastress/>

Tabela 14: Ferramentas para teste de desempenho citadas pelos especialistas.

Identificação do especialista	Quais ferramentas existem para teste de desempenho e o que você sabe sobre elas?
1	JMeter, LoadRunner, Visual Studio e Rational Performance Tester. O JMeter, uma ferramenta open source escrita em Java, possui como foco a execução de testes de desempenho de maneira automatizada. Outras Ferramentas como o LoadRunner da HP e o Rational Performance Tester da IBM possuem algumas características em comum, como por exemplo, utilizar a técnica Record&Playback para a geração de scripts de teste. O Visual Studio da Microsoft possui dois módulos. Um para a criação e execução de cenários de teste com foco em desempenho, são os módulos Web Test e Load Test. O Web Test é o módulo responsável pela criação dos scripts de teste, os quais são gravados utilizando a técnica Record&Playback. Já o módulo Load Test é o módulo responsável pela configuração do cenário de teste.
2	HP LoadRunner - Ferramenta para teste de desempenho top de mercado; suporta a gravação de scripts em diversas linguagens e possui suporte a diversos protocolos de gravação; capaz de configurar cenários de teste simulando comportamento real de usuários interagindo com determinada aplicação a ser testada. MS Visual Studio - suporta a gravação de scripts em C#, Visual Basic; possui suporte a protocolos de gravação de script como HTML/HTTP, porém é inferior a ferramenta LoadRunner neste quesito; capaz de configurar cenários de teste simulando comportamento real de usuários interagindo com determinada aplicação a ser testada.
3	Tenho conhecimento das ferramentas LoadRunner e Visual Studio, que são ferramentas geradoras de scripts de teste. Seu funcionamento é dado pela escrita de ações realizadas por um testador em um script de execução. Este script é utilizado para a geração de um teste de performance que irá simular as mesmas ações sendo tomadas por um número n de usuários.
4	A ferramenta JMeter, que é utilizada para teste de aplicações web.
5	LoadRunner - uso no meu dia-a-dia; JMeter - uso no meu dia-a-dia Visual Studio - uso eventualmente
6	LoadRunner, Visual Studio, JMeter - ambas as ferramentas para teste de sistemas baseados na plataforma web.
7	LoadRunner, Visual Studio e Benchmarks. Possui conhecimento prático sobre a criação e a execução de cenários e testes.
8	LoadRunner: líder de mercado, ótima ferramenta, atende diversos protocolos e multiplataforma, mas sua licença é muito cara. Visual Studio: excelente ferramenta proprietária, módulo de teste embutido na IDE de desenvolvimento do Visual Studio. Entretanto, atende poucos protocolos; funciona somente para plataforma Windows; e, a funcionalidade de geração de relatório/gráficos e análise poderia ser melhor. Apache JMeter: Exclusivo para a linguagem Java, interface não muito amigável, open source. Borland Silk Performer: ferramenta proprietária, funciona somente para plataforma Windows, poderia evoluir em diversas funcionalidades, mas uma boa ferramenta dado seu custo/benefício. The Grinder: Open source, multiplataforma, atende vários protocolos, boa solução para uma ferramenta open source, mas interface não é tão amigável. LoadUI: Open source, multiplataforma, atende vários protocolos, mas possui déficit nos módulos de monitoração e análise dos resultados.

As atividades dos testes de desempenho identificadas pelos especialistas estão listadas na Tabela 15, sendo utilizadas as seguintes referências nesta questão:

- **Participante 1:** J. Meier, C. Farre, P. Bansode, S. Barber e D. Rea. "Performance Testing

Guidance for Web Applications: Patterns & Practices”. Microsoft Press, 2007.

- **Participante 4:** Testar.me consultoria em TI. “Teste de Performance”. Disponível em http://www.testar.me/pages/testar_me_teste_de_performance.html
- **Participante 5:** Testar.me consultoria em TI. “Testes de performance e desempenho de aplicações Web Atividades”. Disponível em http://www.testar.me/pages/teste_de_software_performance_capitulo4.html

Tabela 15: Atividades dos testes de desempenho segundo os especialistas entrevistados.

Identificação do especialista	Quais atividades podem fazer parte de um teste de desempenho?
1	1 - Identificação do ambiente de teste; 2 - Identificação dos critérios de aceitação de desempenho. Quais os objetivos e as restrições de tempo de resposta, vazão e utilização de recursos; 3 - Planejamento e projeto dos testes. Identificar cenários principais, definir os dados do teste e estabelecer métricas para coletar; 4 - Configuração do ambiente de teste. Preparação do ambiente de testes, ferramentas e recursos necessário para executar o teste; 5 - Implementação dos testes; 6 - Execução do teste e monitoramento; 7 - Análise dos resultados.
2	Definição dos requisitos de teste, definição do projeto de teste, definição do plano de teste, definição de scripts e cenário de teste, execução do teste e análise dos resultados do teste.
3	As principais atividades são o levantamento de requisitos não funcionais, o levantamento dos pontos críticos do software (que podem ser afetados por problemas de performance), a geração de scripts de teste a partir das ações de um testador, a execução dos scripts por uma ferramenta adequada e a análise dos resultados desta execução.
4	1 - Identificar o ambiente de teste; 2 - Identificar Desempenho Acceptance Criteria; 3 - Planejar e projetar os testes; 4 - Configurar o ambiente de teste; 5 - Implementar o projeto de teste; 6 – Execução; e 7 - Avaliação.
5	Identificar o ambiente de teste Identificar Desempenho Acceptance Criteria Planejar e projetar os testes Configurar o ambiente de teste Implementar o projeto de teste Execute o teste Resultados analisar, reportar e analisar novamente.
6	Planejamento, programação, execução e análise do teste.
7	Levantamento dos testes, execução e análise dos resultados a fim de identificar gargalos.
8	Analisar os requisitos não funcionais; Avaliar as métricas e contadores; Analisar a documentação do software (e.g. casos de uso); Levantar os perfis dos usuários; Projetar os cenários de testes; Definir os casos/scripts de teste; Planejar a infraestrutura do ambiente do teste de desempenho; Configurar a infraestrutura do ambiente do teste de desempenho; Executar o teste de desempenho; Monitorar o teste de desempenho; Compilar os resultados de desempenho; Analisar os resultados do teste de desempenho; Reportar os resultados do teste de desempenho. Considerando o teste de desempenho baseado em modelos, teríamos algumas atividades extras ou modificadas: Modelar o teste de desempenho; Gerar os cenários de testes baseado no modelo; Gerar os casos/scripts de teste baseado no modelo;

As indicações dos especialistas quanto a como agrupar as atividades dos testes de desempenho estão listadas na Tabela 16. Para responder esta questão nenhum dos participantes utilizou consultas.

Tabela 16: Como agrupar as atividades dos testes de desempenho.

Identificação do especialista	As atividades dos testes de desempenho podem ser agrupadas em diferentes conjuntos? Se sim, quais?
1	Eu acredito que as atividades de teste podem ser classificadas em diferentes conjuntos, pois em cada etapa são realizadas diferentes tarefas, desde a parte de planejamento e preparação, a parte da execução em si e por último a parte da análise dos resultados.
2	<ul style="list-style-type: none"> - Planejamento e definição de requisitos. - Definição e execução dos casos de teste. - Análise dos resultados.
3	Sim. Nomeadas de forma informal, podem ser agrupadas as atividades de levantamento dos testes a serem realizados, geração de scripts, execução do teste e análise de resultados.
4	<ul style="list-style-type: none"> 1 - Identificar o hardware, software e configurações de rede 2 - Diz respeito a identificar o tempo de resposta 3 - Simular variabilidade do teste 4 - Preparar ferramentas para a execução do teste 5 - Desenvolver o projeto de teste 6 - Execução e Avaliação <p>Após executar e avaliar os resultados do teste, se necessário, executá-los novamente.</p>
5	<p>Sim, Requisitos: Identificar o ambiente de teste e Identificar Desempenho</p> <p>Acceptance Criteria Definição dos casos de teste: Planejar e projetar os testes</p> <p>Execução: Configurar o ambiente de teste, Implementar o projeto de teste e Execução do teste</p> <p>Resultados: analisar, reportar e analisar novamente.</p>
6	Planejamento, Programação, Execução e Análise.
7	De forma macro as atividades podem ser agrupadas em levantamento dos testes, execução e análise dos resultados a fim de identificar gargalos.
8	<p>Talvez sim, depende do nível de granularidade:</p> <ul style="list-style-type: none"> * Analisar os requisitos não funcionais; * Avaliar as métricas e contadores; * Analisar a documentação do software (e.g. casos de uso); * Projetar os cenários de testes + Levantar os perfis dos usuários; * Definir os casos/scripts de teste; * Planejar + Configurar a infraestrutura do ambiente do teste de desempenho; * Executar + Monitorar o testes de desempenho; * Compilar + Analisar + Reportar os resultados dos testes de desempenho;

As respostas da pergunta “Quais artefatos (ou entregáveis) podem resultar de um teste de desempenho?” estão apresentadas na Tabela 17. Apenas o participante 4 utilizou consulta nesta questão, que foi a seguinte:

- **Participante 4:** Rational Software Corp. “Rational Unified Process: Artefatos”. Disponível em http://www.wthreex.com/rup/process/artifact/ovu_arts.htm.

Tabela 17: Entregáveis que podem resultar de um teste de desempenho.

Identificação do especialista	Quais artefatos (ou entregáveis) podem resultar de um teste de desempenho?
1	Demonstra as características necessárias que o sistema deve possuir para atender a demanda. Quais os recursos físicos necessários e como deve ser realizado o gerenciamento das transações.
2	É possível entregar documentos com informações/dados referentes aos resultados da execução do teste de desempenho. Posteriormente, é possível correlacionar dados de determinadas métricas e analisar o comportamento da aplicação.
3	Um script de teste de desempenho, uma análise dos resultados e possivelmente um documento de sugestões de correção.
4	Documento, Modelo, Elemento do Modelo.
5	Requisitos, plano e resultados dos testes.
6	Documentos contendo as métricas testadas e a relação entre o desempenho esperado e o desempenho medido.
7	Logs, traces e relatórios.
8	<ul style="list-style-type: none"> * Plano do teste de desempenho * Modelo de teste de desempenho (abordagem MBT) * Cenários e Scripts/Casos de teste de desempenho (código fonte) * Resultados da execução (output da ferramenta) * Relatório do teste de desempenho

A Tabela 18 contém as respostas dos especialistas para a pergunta “Quais métricas de desempenho podem ser monitoradas”. Para responder esta questão as seguintes referências foram utilizadas:

- **Participante 3:** Testar.me consultoria em TI. “Teste de Performance”. Disponível em http://www.testar.me/pages/testar_me_teste_de_performance.html.
- **Participante 4:** J. Myerson. “Crie uma política de métricas de desempenho da nuvem”. Disponível em <http://www.ibm.com/developerworks/br/cloud/library/cl-cloudperformmetrics/>.

Tabela 18: Métricas de desempenho que podem ser monitoradas em um teste.

Identificação do especialista	Quais métricas de desempenho podem ser monitoradas?
1	Speedup, Eficiência, Utilização, Escalabilidade, Tempo de resposta
2	Tempo de resposta, Transações por segundo (TPS), Utilização de CPU, Utilização de memória, Taxa de dados trafegados na rede, Utilização do disco
3	As métricas do teste de desempenho são principalmente relacionadas ao uso de recursos computacionais (memória, processador) e tempo de execução de tarefas conforme aumento de usuários.
4	1 – statefulness; 2 – versão; 3 - limite de recurso; 4 - limite de usuário; 5 - limite de solicitação de dados; 6 - limite de resposta.
5	CPU, memória, vazão, tempo de resposta, e/s,
6	Velocidades e taxas de acesso e transferência de informações (disco, rede, processador, memória, etc);
7	Software ou hardware tais como: bytes/s e tempo de resposta.
8	<p>WINDOWS: Request response time; Transaction response time; Requests/sec; Transaction/sec; Requests execution time; % Processor time; % Privileged time; % Interrupt time; Processor Queue Length; Context Switches; System Up Time; Available Mbytes; Working Set; Pages/sec; Page Read/sec; Pool Nonpaged Bytes; Paged Pool Bytes; Paged Pool Failures; Cache Faults/sec; Cache Bytes; System Cache Resident Bytes; Committed Bytes; Avg. Disk secs/transfer; % Idle Time; Disk Transfer/sec; Avg. Disk Queue Length; Split IO/sec; Free Megabytes; Page faults/sec; Working set; Private Bytes; Handle Count; Bytes Total/sec; Server Bytes Total; Datagrams/sec; Connections Established; Segments Received/sec; % Interrupt Time</p> <p>UNIX: CPU utilization; User mode CPU utilization; System mode CPU utilization; Average Load; Interrupt rate; Context switches rate; Percent Used Memory; MB Free; Paging Rate; Page-in Rate; Page-out Rate; IO % Used; Free; Disk Rate; Incoming packets rate; Outcoming packets rate; Incoming packets error; Outcoming packets error; Collision rate.</p>

A opinião dos especialistas em teste de desempenho sobre como agrupar as métricas de desempenho pode ser encontrada na Tabela 19. Para responder esta questão as seguintes referências foram utilizadas:

- **Participante 4:** J. Myerson. “Crie uma política de métricas de desempenho da nuvem”. Disponível em <http://www.ibm.com/developerworks/br/cloud/library/cl-cloudperformmetrics/>.
- **Participante 8:** Hewlett-Packard. “HP Performance Engineering Best Practices Series - Performance Monitoring Best Practices”. 2010.

Tabela 19: Como agrupar as métricas de desempenho de acordo com os especialistas.

Identificação do especialista	As métricas de desempenho podem ser agrupadas em diferentes conjuntos? Se sim, quais?
1	Não imagino uma forma de agrupar as métricas de desempenho.
2	- Recursos computacionais - Métricas de comportamento do usuário ao interagir com a aplicação
3	Sim, métricas de desempenho físico e de tempo de execução.
4	1 - Refere-se a quão bem o aplicativo responde. 2 - Quão bem uma nova compilação evita estragar as funções existentes. 3 - Quão bem o consumo de recursos é equilibrado dinamicamente para aplicativos. 4 - Como a aplicação responde a vários usuários simultaneamente. 5 - solicitações de dados que podem ser processadas rapidamente. 6 - Quanto tempo o aplicativo leva para responder.
5	As métricas podem ser agrupadas em categorias como: rede, recursos, Entrada/Saída.
6	Processador: uso de CPU, etc. Memória: velocidade de acesso, uso, eficiência, média de tempo, média de uso, etc. Disco: velocidade de leitura e escrita, eficiência, etc. Rede: Tráfego de rede (entrada / saída), pacotes p/ minuto, acesso simultâneo, etc.
7	Operacionais e derivadas.
8	Tempo de Resposta (Response Time), Vazão (Throughput), Utilização de Recursos (CPU, Memória, Disco E/S, Rede E/S, Processos).

Os participantes foram perguntados o quanto estavam de acordo que um teste de desempenho possa possuir os objetivos listados a seguir e as respostas podem ser encontradas na Tabela 20.

- O1. Avaliação do desempenho sob condições normais de carga.
- O2. Avaliação do desempenho durante longos períodos de interação.
- O3. Comparação do desempenho em diferentes configurações.
- O4. Elaboração de um entregável de teste de desempenho.
- O5. Identificação de gargalos de desempenho.
- O6. Melhoria do desempenho do sistema.
- O7. Verificação dos requisitos de desempenho.

Quando perguntados se adicionariam algum novo objetivo ou removeriam algum dos objetivos citados, apenas o participante número 8 respondeu que removeria a elaboração de um entregável de teste de desempenho. Este mesmo participante adicionaria:

- Validação dos requisitos não funcionais especificados.
- Avaliação do desempenho em condições elevadas de carga.
- Determinar os limites de processamento do sistema.
- Verificar se o desempenho do software satisfaz os requisitos especificados.

Tabela 20: Nível de concordância dos especialistas em relação aos objetivos representados na ontologia.

Identificação do especialista	O1	O2	O3	O4	O5	O6	O7
1	4	4	4	4	4	4	4
2	4	4	4	4	4	4	4
3	4	5	5	5	4	5	4
4	4	4	4	3	4	4	2
5	5	5	5	5	5	5	5
6	5	5	4	4	4	3	3
7	4	5	4	3	4	2	4
8	5	5	5	2	5	4	5
Média	4.37	4.62	4.37	3.75	4.25	3.87	3.87

As subclasses do conceito *PTArtifact* também foram apresentadas aos participantes. Foi perguntado o quanto eles estão de acordo que um teste de desempenho possa resultar nos entregáveis enumerados a seguir e as respostas obtidas estão na Tabela 21:

- E1. Caso de teste de desempenho.
- E2. Plano de teste de desempenho.
- E3. Medições de desempenho.
- E4. Relatório do teste de desempenho.
- E5. Relatório de incidente ocorrido em um teste desempenho.
- E6. Registro (Log) do teste de desempenho.
- E7. Especificação do caso de teste de desempenho.
- E8. Especificação do procedimento de teste de desempenho.
- E9. Especificação do projeto de teste de desempenho.

Tabela 21: Nível de concordância dos especialistas em relação aos entregáveis representados na ontologia.

Identificação do especialista	E1	E2	E3	E4	E5	E6	E7	E8	E9
1	4	4	4	4	4	4	4	4	4
2	2	2	4	4	4	4	4	4	4
3	5	5	4	4	5	5	4	4	4
4	3	3	5	5	5	4	3	3	2
5	5	5	5	5	3	2	3	3	3
6	4	4	4	3	3	2	3	3	4
7	4	4	2	4	3	4	4	3	4
8	5	5	4	5	4	5	4	4	4
Média	4	4	4	4.25	3.87	3.75	3.62	3.5	3.62

Quando perguntados se adicionariam algum novo entregável ou removeriam algum dos entregáveis citados, o participante número 8 comentou pensar que os itens “Especificação do procedimento de teste de desempenho” e “Especificação do projeto de teste de desempenho” são subitens do “Plano de teste de desempenho”. Além disso, a “Especificação do caso de teste de desempenho” é subitem de “Caso de teste de desempenho” ou ele não entendeu a diferença. Ainda, o participante número 8 adicionaria o item “Cenários de teste de desempenho”.

Os conceitos que pertencem à hierarquia de *PTActivity*, listados a seguir, também foram apresentados aos participantes. Foi perguntado o quanto os especialistas estão de acordo que um teste de desempenho possa possuir estas atividades e os resultados podem ser encontrados divididos entre a Tabela 22 e a Tabela 23.

- A1. Análise das medições de desempenho
- A2. Definição dos requisitos de desempenho
- A3. Desenvolvimento dos casos de teste de desempenho
- A4. Parametrização dos casos de teste de desempenho
- A5. Elaboração do plano do teste de desempenho
- A6. Elaboração do relatório do teste de desempenho
- A7. Configuração do ambiente de teste
- A8. Execução dos casos de teste de desempenho

Tabela 22: Nível de concordância dos especialistas em relação às atividades da ontologia (parte 1).

Identificação do especialista	A1	A2	A3	A4	A5	A6	A7	A8
1	4	4	3	4	4	4	4	4
2	4	4	4	4	4	4	4	4
3	4	4	4	4	4	4	5	4
4	4	3	3	4	3	2	4	4
5	5	5	5	5	5	5	3	5
6	4	5	4	4	5	3	5	3
7	4	3	4	4	4	4	4	4
8	4	4	4	4	4	4	4	4
Média	4.12	4	3.87	4.12	4.12	3.75	4.12	4

Na Tabela 23 foi utilizada a seguinte legenda para as atividades:

- A9. Monitoramento das métricas de desempenho
- A10. Definição das atividades do teste

- A11. Definição da abordagem do teste
- A12. Definição dos parâmetros do caso de teste
- A13. Definição dos objetivos do teste
- A14. Definição do ambiente de teste
- A15. Definição dos casos de uso do teste
- A16. Definição da carga

Tabela 23: Nível de concordância dos especialistas em relação às atividades da ontologia (parte 2).

Identificação do especialista	A9	A10	A11	A12	A13	A14	A15	A16
1	4	4	4	4	4	4	4	4
2	4	4	4	4	4	4	4	4
3	4	4	4	4	5	5	4	4
4	4	3	2	2	4	4	4	3
5	5	5	5	4	4	4	5	5
6	5	5	4	4	4	5	4	4
7	4	3	3	4	4	4	4	4
8	4	2	3	3	3	3	3	3
Média	4.25	3.75	3.62	3.62	4	4.12	4	3.87

Quando perguntados se adicionariam alguma nova atividade ou removeriam alguma das atividades citadas, apenas o participante número 8 respondeu que é possível remover ou agrupar alguns dos conceitos, pois ele pensa que os itens a seguir estão relacionados:

- Definição da abordagem do teste → Elaboração do plano do teste de desempenho;
- Definição dos parâmetros do caso de teste → Parametrização dos casos de teste;
- Definição dos objetivos do teste → Elaboração do plano do teste de desempenho;
- Definição do ambiente de teste → Configuração do ambiente de teste;
- Definição dos casos de uso do teste → Desenvolvimento dos casos de teste;
- Definição da carga → Elaboração do plano do teste de desempenho

Os participantes também foram perguntados o quanto estavam de acordo em agrupar as atividades dos testes de desempenho nas seguintes categorias e as respostas podem ser visualizadas na Tabela 24:

- C1. Atividades de planejamento
- C2. Atividades de desenvolvimento
- C3. Atividades de documentação
- C4. Atividades de execução

Ainda referente as atividades de teste, quando perguntados se adicionariam alguma nova categoria ou removeriam alguma das categoria citadas, os participantes número 2 e 5 coincidentemente apresentaram a mesma resposta: eles adicionariam uma categoria denominada “atividades de análise”.

Tabela 24: Nível de concordância dos especialistas em relação ao agrupamento das atividades na ontologia.

Identificação do especialista	C1	C2	C3	C4
1	4	4	4	4
2	4	4	4	4
3	5	5	5	5
4	5	4	5	5
5	5	5	5	5
6	3	4	2	4
7	5	5	5	5
8	5	5	5	5
Média	4.5	4.5	4.37	4.62

Como pode ser visualizados na Tabela 25, apenas 3 dos 8 participantes registraram comentários adicionais sobre o questionário.

Tabela 25: Comentários adicionais dos participantes sobre as perguntas abertas.

Identificação do especialista	Comentário adicional sobre as perguntas abertas
2	Questionário com certa complexidade para quem não possui experiência. A utilização da web para obter as respostas do survey pode não ser uma opção adequada, pois o teste de desempenho possui diferentes definições e algumas vezes pode confundir o entendimento.
3	A grande maioria das questões não foi respondida usando consulta para representar meu conhecimento adquirido trabalhando na área.
8	Foi demorado para responder ao questionário, poderia ter algumas opções já configuradas, mas entendo o objetivo do questionário. Penso que um teste que aplique uma abordagem com ou sem uso de modelos interfere no processo. Nas atividades seria interessante questionar a ordem (precedência) entre as atividades.

APÊNDICE C – COMPARAÇÃO DO QUESTIONÁRIO COM A ONTOLOGIA

Este apêndice tem como finalidade comparar as respostas dos especialistas com o conhecimento representado na ontologia proposta. A Tabela 26 compara as semelhanças e as diferenças entre as respostas dos especialistas e o conhecimento representado na ontologia quanto aos objetivos dos testes de desempenho. Os conceitos que representam os objetivos dos testes foram abreviados da seguinte forma:

- O1. Avaliação do desempenho sob condições normais de carga
- O2. Avaliação do desempenho durante longos períodos de interação
- O3. Comparação do desempenho em diferentes configurações
- O4. Elaboração de um entregável de teste de desempenho
- O5. Identificação de gargalos de desempenho
- O6. Melhoria do desempenho do sistema
- O7. Verificação dos requisitos de desempenho

Tabela 26: Semelhanças e diferenças entre os objetivos dos especialistas e os objetivos da ontologia proposta.

Identificação do especialista	Semelhanças		Diferenças		Total de respostas
	Total	Conceitos	Total	Objetivo	
1	3	O1, O5, O6	1	Estimar uma configuração de hardware adequada para cada aplicação.	4
2	1	O7	0	-	1
3	1	O6	0	-	1
4	1	O3	0	-	1
5	2	O1, O7	0	-	2
6	1	O1	0	-	1
7	2	O1, O5	0	-	2
8	2	O6, O7	0	-	2
Total	13		1		14
Média	1.62		0.12		1.75

A Figura 56 ilustra os objetivos dos testes de desempenho que foram citados com maior frequência. É importante citar que o objetivo mais citado (por 4 dos 8 especialistas) é a “avaliação do desempenho sob condições normais de carga”. Contudo, os conceitos O2 e O4 não foram identificados pelos especialistas.

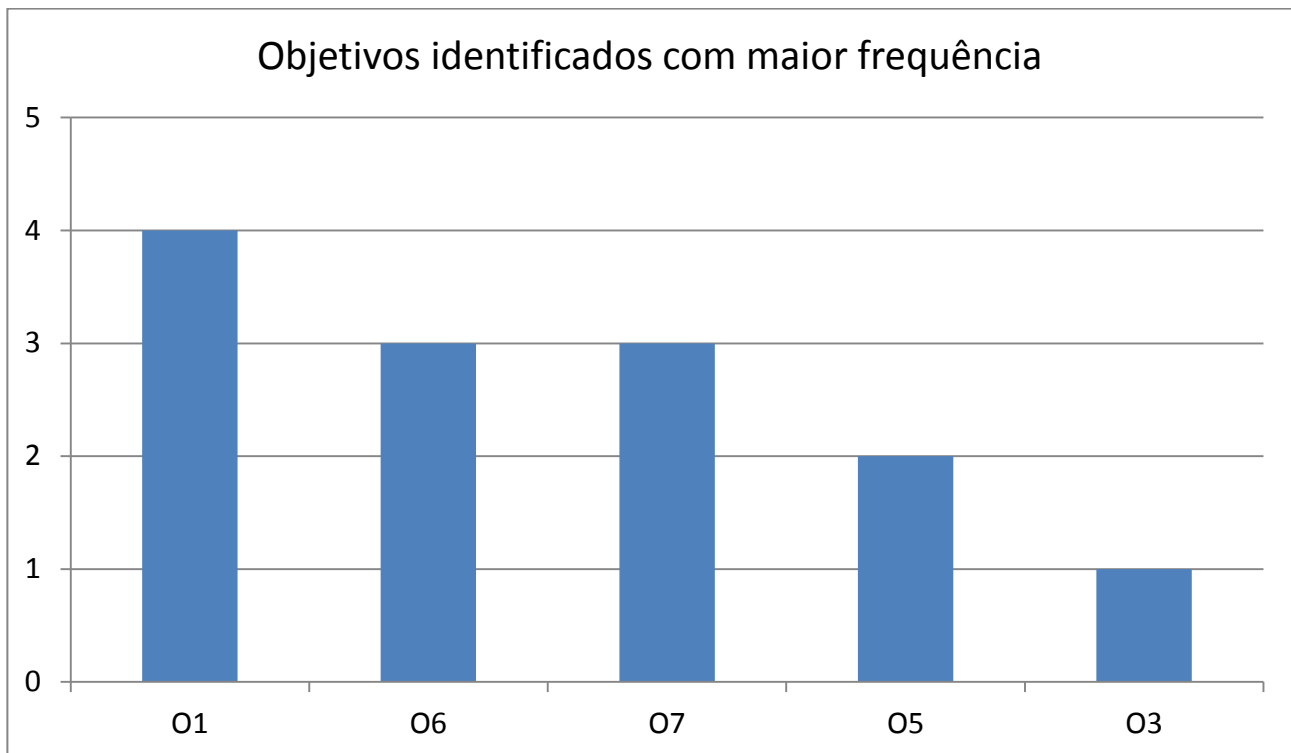


Figura 56: Objetivos dos testes de desempenho mais frequentemente citados pelos especialistas.

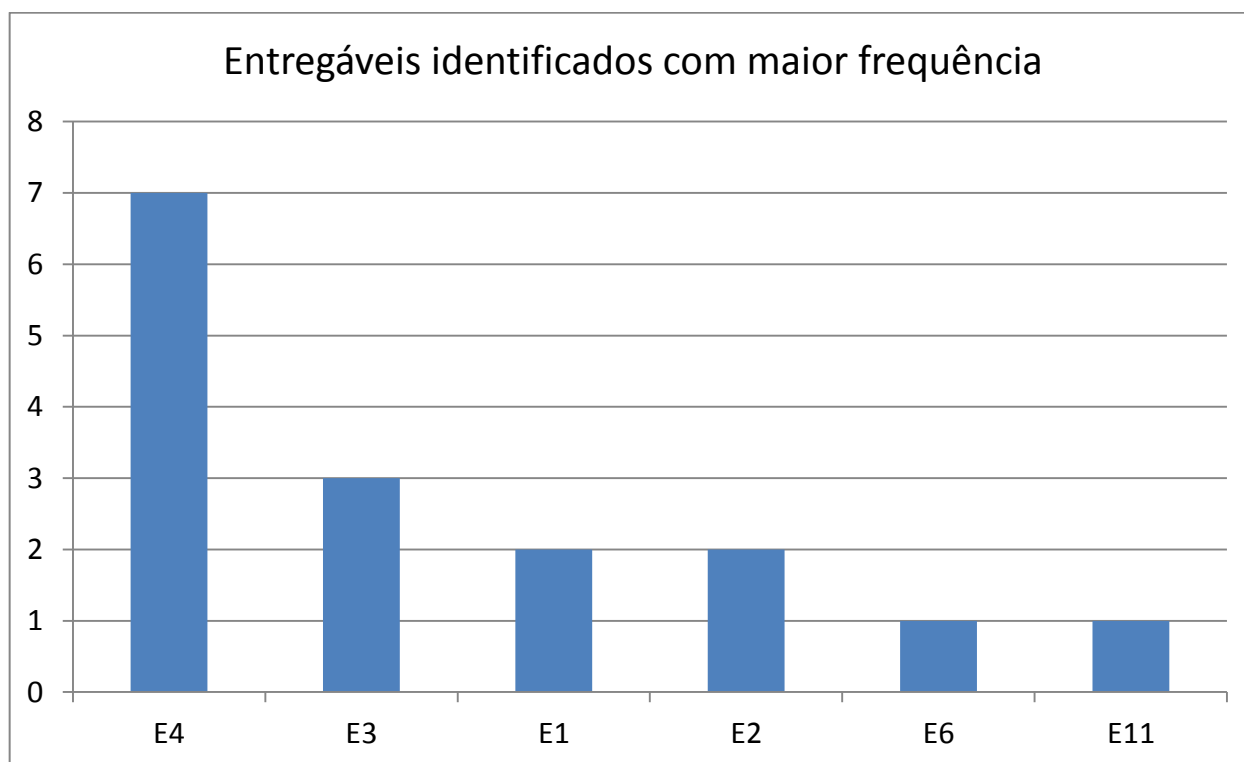
A Tabela 27 compara as semelhanças e as diferenças entre as respostas dos especialistas e o conhecimento representado na ontologia quanto aos artefatos (também chamados entregáveis) dos testes de desempenho. Uma diferença que foi citada por 2 especialistas são os modelos dos teste de desempenho, caso o teste venha a utilizar uma abordagem baseada em modelos. Os conceitos que representam os entregáveis dos testes foram abreviados da seguinte forma:

- E1. Caso de teste de desempenho
- E2. Plano de teste de desempenho
- E3. Medições de desempenho
- E4. Relatório do teste de desempenho
- E5. Relatório de incidente ocorrido no teste desempenho
- E6. Registro (Log) do teste de desempenho
- E7. Especificação do caso de teste de desempenho
- E8. Especificação do procedimento de teste de desempenho
- E9. Especificação do projeto de teste de desempenho
- E10. Relatório de transmissão de item de teste
- E11. Documento de teste

Tabela 27: Semelhanças e diferenças entre os entregáveis dos especialistas e os existentes na ontologia.

Identificação do especialista	Semelhanças		Diferenças		Total de respostas
	Total	Conceitos	Total	Entregável	
1	1	E4	0	-	1
2	2	E3, E4	0	-	2
3	2	E1, E4	0	-	2
4	1	E11	2	Modelo e elemento do modelo.	3
5	2	E2, E4	1	Requisitos.	3
6	2	E3, E4	0	-	2
7	2	E4, E6	1	Traces.	3
8	4	E1, E2, E3, E4	1	Modelo do teste de desempenho.	5
Total	16		5		21
Média	2		0.62		2.62

A Figura 57 ilustra a frequência com que os artefatos foram citados pelos especialistas. Podemos perceber que o “relatório do teste de desempenho” foi lembrado por 7 dos 8 participantes. Contudo, os conceitos E5, E7, E8, E9 e E10 não foram citados nas respostas dos questionários.

**Figura 57:** Artefatos dos testes de desempenho mais frequentemente citados pelos especialistas.

A Tabela 29 compara as semelhanças e as diferenças entre as respostas dos especialistas e o conhecimento representado na ontologia quanto às ferramentas dos testes de desempenho. Em média, os participantes enumeraram 4.75 ferramentas, dos quais 2.87 se referem a instâncias e 1.88 são conceitos que especializam *PTTool*. As instâncias das ferramentas de testes citadas foram abreviadas da seguinte forma:

- F1. JMeter
- F2. LoadRunner
- F3. Rational Performance Tester
- F4. VisualStudio

Além disso, os seguintes conceitos que especializam *PTTool* foram citados:

- C1. Ferramenta para geração de carga
- C2. Ferramenta para teste HTTP
- C3. Ferramenta de monitoração
- C4. Ferramenta disponível para o Windows

Tabela 28: Semelhanças e diferenças entre as ferramentas dos especialistas e as ferramentas da ontologia.

Identificação do especialista	Semelhanças		Diferenças		Total de respostas
	Total	Conceitos ou instâncias	Total	Ferramenta	
1	5	F1, F2, F3, F4, C1	2	Ferramenta open source, Ferramenta para geração de teste	7
2	3	F2, F4, C1	1	Ferramenta para geração de teste	4
3	3	F2, F4, C1	1	Ferramenta para geração de teste	4
4	1	F1	0	-	1
5	3	F1, F2, F4	0	-	3
6	4	F1, F2, F4, C2	0	-	4
7	3	F2, F4, C1	1	Ferramenta para geração de teste	4
8	5	F1, F2, F4, C3, C4	6	loadUI, The Grinder, Borland Silk Performer, Ferramenta de análise, open source ou proprietária	11
Total	27		11		38
Média total	3.37		1.37		4.75
Média conceitos	0.88		1.00		1.88
Média instâncias	2.50		0.37		2.87

A Figura 58 ilustra a frequência com que as instâncias ou conceitos de ferramentas de teste foram citadas pelos especialistas. O Visual Studio e o LoadRunner foram as

ferramentas mais citadas, por 7 dos 8 participantes. Em segundo lugar, o JMeter foi citado por 5 especialistas. Contudo, apesar da ontologia possuir informações sobre 18 instâncias do conceito *PTTool*, apenas 4 ferramentas foram citadas pelos participantes. O conceito mais citado da ontologia, por 4 participantes, foram as ferramentas para geração de carga. Por outro lado, 4 participantes citaram um conceito que ainda não existe na ontologia, que é “ferramenta para geração de teste”. Este conceito pode ser criado como qualquer ferramenta de teste (subclasse de *PTTool*) que suporta (propriedade *supports*) a atividade de desenvolvimento dos casos de teste (conceito *PTCaseDevelopment*).

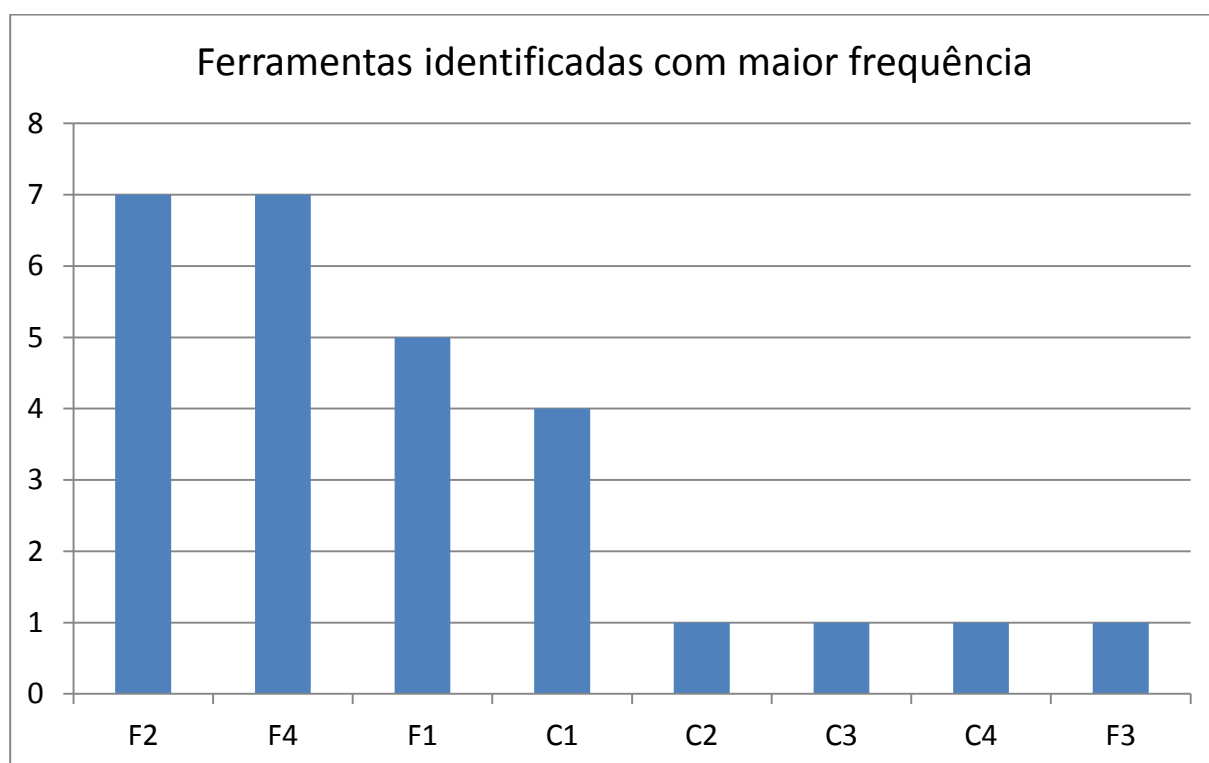


Figura 58: Ferramentas de teste de desempenho identificadas pelos especialistas.

A Tabela 29 compara as semelhanças e as diferenças entre as respostas dos especialistas e o conhecimento representado na ontologia quanto às atividades dos testes de desempenho. Em média, os participantes enumeraram 6.87 atividades, das quais 6.25 já estão representadas na ontologia. Os conceitos que representam as atividades dos testes foram abreviados da seguinte forma:

- A1. Análise das medições de desempenho
- A2. Definição dos requisitos de desempenho
- A3. Desenvolvimento dos casos de teste de desempenho
- A4. Parametrização dos casos de teste de desempenho

- A5. Elaboração do plano do teste de desempenho
- A6. Elaboração do relatório do teste de desempenho
- A7. Configuração do ambiente de teste
- A8. Execução dos casos de teste desempenho
- A9. Monitoramento das métricas de desempenho
- A10. Definição das atividades do teste
- A11. Definição da abordagem do teste
- A12. Definição dos parâmetros do caso de teste
- A13. Definição dos objetivos do teste
- A14. Definição do ambiente de teste
- A15. Definição dos casos de uso do teste
- A16. Definição da carga
- A17. Definição das métricas a serem monitoradas.

Tabela 29: Semelhanças e diferenças entre as atividades dos especialistas e as atividades da ontologia.

Identificação do especialista	Semelhanças		Diferenças		Total de respostas
	Total	Conceitos	Total	Atividade	
1	11	A1, A2, A3, A5, A7, A8, A9, A12, A14, A16, A17	0	-	11
2	5	A1, A2, A3, A5, A8	1	Definição do projeto de teste	6
3	4	A1, A2, A3, A8	1	Levantamento dos pontos críticos do software	5
4	7	A1, A2, A3, A5, A7, A8, A14	0	-	7
5	8	A1, A2, A3, A5, A6, A7, A8, A14	0	-	8
6	4	A1, A3, A5, A8	0	-	4
7	3	A1, A5, A8	0	-	3
8	8	A1, A6, A7, A8, A9, A14, A15, A16	3	Analisar os requisitos não funcionais, modelar o teste e gerar os cenários/casos/scripts de testes baseado no modelo.	11
Total	50		5		55
Média	6.25		0.62		6.87

A Figura 59 ilustra a frequência com que as atividades foram citadas pelos especialistas. Podemos perceber que todos os 8 participantes foram unânimes em

responder as atividades de “análise das medições de desempenho” e “execução dos casos de teste”. Em segundo lugar, com 6 votos, empatam as atividades de “desenvolvimento dos casos de teste” e “elaboração do plano de teste”. Contudo, os conceitos A4, A10, A11 e A13 não foram citados nas respostas dos questionários.

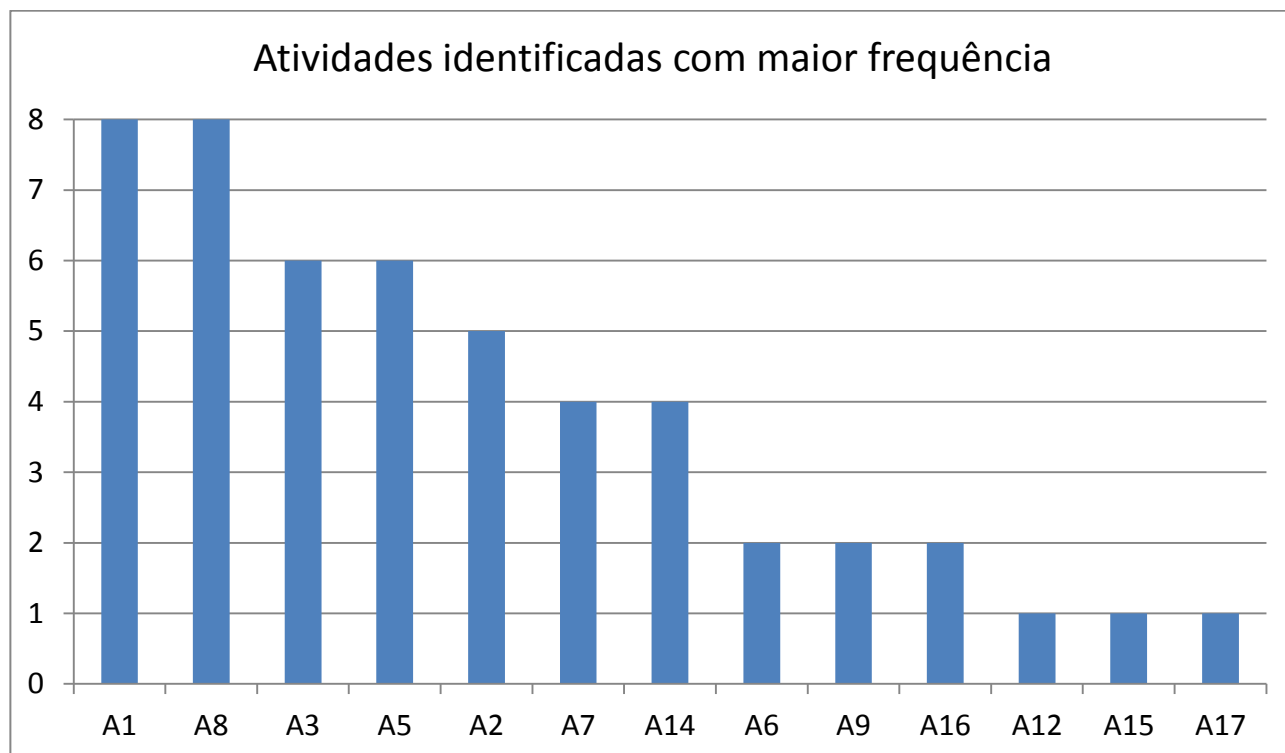


Figura 59: Atividades dos testes de desempenho identificadas pelos especialistas.

A Tabela 30 compara as semelhanças e as diferenças entre as respostas dos especialistas e o conhecimento representado na ontologia quanto às métricas de desempenho. Em média, os participantes enumeraram 11.75 métricas de desempenho, das quais 7.25 são conceitos e 4.50 são instâncias. A ontologia possui 23 conceitos do tipo *PerformanceMetric* e 107 instâncias relacionadas. Os conceitos que representam as métricas de desempenho foram abreviados da seguinte forma:

- M1. Métricas de utilização dos recursos computacionais
- M2. Métricas devido à interação com a aplicação
- M3. Métricas de memória
- M4. Métricas de processador
- M5. Métricas de rede
- M6. Métricas de disco
- M7. Métricas de processo

As instâncias de métricas de desempenho foram abreviadas da seguinte forma:

- I1. Tempo de resposta
- I2. Transações por segundo
- I3. Utilização da CPU
- I4. Utilização da memória
- I5. Utilização do disco
- I6. Bytes trafegados na interface de rede por segundo
- I7. Número de leituras em disco por segundo
- I8. Número de escritas em disco por segundo
- I9. Pacotes enviados por segundo na interface de rede
- I10. Pacotes recebidos por segundo na interface de rede

Tabela 30: Semelhanças e diferenças entre as métricas dos especialistas e as métricas da ontologia proposta.

Identificação do especialista	Semelhanças		Diferenças		Total de respostas
	Total	Conceitos ou Instâncias	Total	Métricas	
1	3	M1, M2, I1	3	Speedup, eficiência, escalabilidade	6
2	12	M1, M2, M3, M4, M5, M6, I1, I2, I3, I4, I5, I6	0	-	12
3	5	M1, M2, M3, M4, I1	1	Número de usuários	6
4	4	M1, M2, I1, I6	3	Statefulness, versão, limite de usuário	7
5	7	M1, M3, M4, M5, M6, I1, I6	0	-	7
6	13	M1, M3, M4, M5, M6, I3, I4, I5, I6, I7, I8, I9, I10	0	-	13
7	3	M1, M2, I1	0	-	3
8	34	Todos os conceitos e as instâncias anteriormente listados foram citados e mais as respostas abaixo*	6	Paged pool failures, system cache resident bytes, collision rate, system up time, disk Transfer/sec, avg disk secs/transfer.	40
Total	81		13		94
Média total	10.12		1.62		11.75
Média conceitos	4.12		0.37		4.50
Média instâncias	6.00		1.25		7.25

*O participante 8 identificou um número maior de métricas que existiam na ontologia, que foram as seguintes: Requisições por segundo; Porcentagem do tempo de CPU em modo privilegiado; Porcentagem do tempo de CPU tratando interrupções; Fila de processos no processador; Trocas de contexto por segundo; Páginas escritas em disco ou lidas do disco por segundo (pages per second); Leitura de páginas do disco por segundo; Alocações de pool não-pagináveis em bytes; Alocações de pool pagináveis em bytes; Falhas na memória cache por segundo; Bytes de cache; Falhas na memória por segundo; Bytes confirmados; Tamanho médio da fila de disco; Datagramas UDP por segundo; Segmentos TCP recebidos por segundo; e Conexões TCP estabelecidas.

A Figura 60 ilustra as métricas de desempenho que foram citadas com maior frequência pelos especialistas. Podemos perceber que todos os 8 participantes citaram o conceito *MachinePerformanceMetric* e 7 identificaram a instância que representa a métrica “tempo de resposta”.

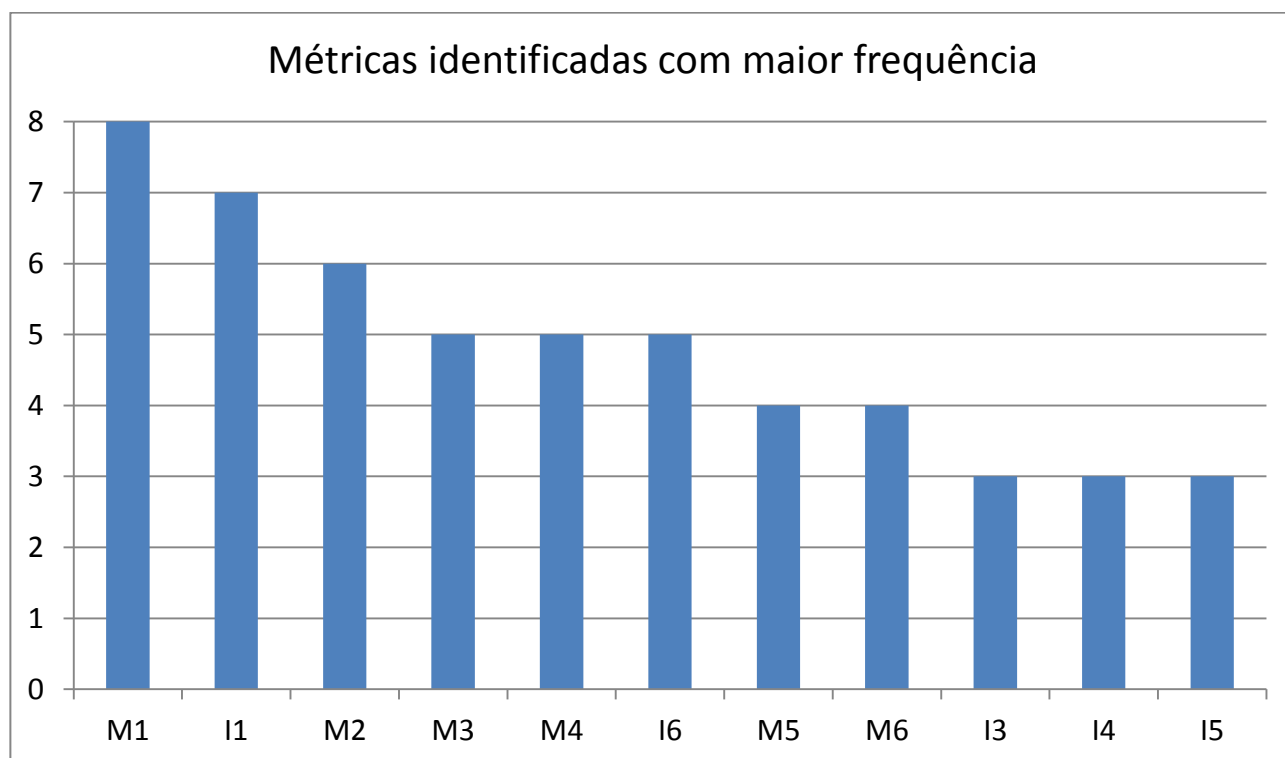


Figura 60: Métricas de desempenho identificadas com maior frequência pelos especialistas.