An Ontology for Guiding Performance Testing

Artur Freitas and Renata Vieira
Postgraduate Program in Computer Science – Faculty of Informatics (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, RS – Brazil
artur.freitas@acad.pucrs.br, renata.vieira@pucrs.br

***Abstract*—Software test is a technique to obtain information about software systems quality. Performance test is a type of software test that aims at evaluating software performance at a given load scenario, but it requires specialized knowledge about tools, activities and metrics of the domain. Since ontology is a promising knowledge representation technique, this paper presents a literature review to identify trends and compare researches of ontologies in the fields of software testing and software performance. Also, to investigate this issue from a practical perspective, it was developed an ontology for representing the core knowledge of performance testing. This paper presents the ontology and compare it with related ones. Then, semantic technologies are explored to demonstrate the practical feasibility of developing ontology-based applications for assisting testers with performance test planning and management.**

I. Introduction

Planning and executing software tests demand specialized knowledge about testing techniques, criteria, artifacts and tools, among others. This diversity of concepts and their relationships make the establishment of a common understanding an issue to be considered. Ontology is a technique for capturing domain knowledge and, therefore, it shows great potential to be used in the knowledge rich testing process. Ontologies have achieved an important role with respect to information systems, knowledge management and information sharing systems as well as for the development of fields such as semantic technologies. On software testing, ontologies can provide a precise choice of terms to be used in the communication between testers, developers, managers and users. Thus, ontologies make explicit the hidden assumptions related to this practice [1], and support the acquisition, organization, reuse and sharing of knowledge in the domain. This work reports both theoretical findings and practical experiences obtained to answer the following research questions: **RQ1.** What is the state of the art of using ontologies in software testing? **RQ2.** Can ontologies represent performance testing knowledge? **RQ3.** Which inferences are enabled by a performance testing ontology?

To answer the **RQ1**, papers using ontologies for software testing activities were analysed and compared based on several criteria such as, if they propose new ontologies, present ontology-based applications, use ontology technologies, and compare or evaluate ontologies. This field mapping guided the decisions in the remainder of the research, *i.e.*, the most used ontology technologies were also adopted to build our ontology of performance testing. To investigate the **RQ2**, we built the performance testing ontology in OWL [2] (Web Ontology Language) with Protégé [3] and compare it with the two most related ontologies. To evaluate the **RQ3**, ontology-based applications were developed employing semantic technologies, such as Pellet [4], SQRWL [5], OWL API [6] and Protégé [3]. At a higher level of abstraction, the applications help testers to define what have to be taken into consideration for the execution of performance tests. The ontology-based applications aim to help performance testers to manage instances of concepts and relationships in this domain, such as performance test tools, activities, metrics, goals and artifacts. The ontology is applied to support tester's decisions by suggesting, for example, a methodology for test development or indicating tools aligned with the test goals. Then, it is possible to generate a performance test plan according with testers decisions.

This paper is organized as follows. Section II explains our literature review about ontologies in software testing and software performance. Section III shows the scope and conceptualization of our proposed performance testing domain ontology. Section IV presents an evaluation of this ontology by comparing its metrics with related ones. Section V explains ontology-based applications developed based on our ontology. Section VI concludes this research.

II. Literature Review

Ontology is a technique to represent and manipulate knowledge. To clarify the main researches and to identify the state of the art regarding the use of ontologies in the domains of software testing and software performance, a literature review was conducted. By mapping the field, it becomes possible to identify research trends, implications for practice, open issues and areas for improvement. Since it was not found any systematic review or systematic mapping with the same scope, this literature review is an opportunity to bring interesting scientific findings. The scope of the review embraces researches that study or apply ontologies to benefit the domains of software testing or software performance. Therefore, its main objective is to assess the impact and provide a comprehensive overview about the use of ontologies in the mentioned areas. Following the systematic review vocabulary [7] this literature review has as **population** software professionals related with software testing or software performance; as **intervention** any ontology-based approach; and as **outcome** any type of evidence about the use of ontologies on software testing or software performance. To accomplish this review, three article databases[1] were consulted: IEEE Xplore Digital Library, Scopus and Compendex. The search string adopted is the following:

ontology AND ("software testing" OR "software test" OR "software performance" OR "system performance" OR "performance testing" OR "performance test" OR "load test" OR "load testing")

[1]Databases: IEEE Xplore Digital Library http://ieeexplore.ieee.org/, Scopus http://www.scopus.com/, and Compendex http://www.engineeringvillage.com/

978-1-4799-4143-8/14 $31.00 © 2014 IEEE
DOI 10.1109/WI-IAT.2014.62

This review was conducted in March 2013 and it was searched all articles up to 2012 (inclusive). Table I shows the results obtained by using the search string in the aforementioned databases. The search on Scopus was limited to article tittle, abstract and keywords, and from its results 20 papers were filtered because they are conference reviews. The reason to remove 5 results from the IEEExplore is that 4 are conference proceedings and 1 result came duplicated. From the Compendex results, 11 papers were excluded since they are conference proceedings (not journal or conference article). From this total of 314 articles returned in the databases, 79 were removed because they are duplicated results retrieved by more than one database, resulting in 235 different papers. For the next step, the title, abstract and keywords of these 235 resulting articles were fully read to analyse if each paper meets the inclusion criterion of using ontologies in the fields of software testing or software performance. More precisely, a work that makes at least one of the following contributions was considered likely to be relevant to the field:

IC1. Propose an ontology related with the domain of software testing or software performance.
IC2. Present an approach or application based on ontologies to improve software testing or software performance.
IC3. Evaluate or compare ontologies related with software testing or software performance.

Therefore, a paper is excluded from this literature review if it does not present at least one of the inclusion criteria defined above. Likewise, exclusion criteria were defined to complement those inclusion criteria by defining characteristics that classify a paper as out of the desired scope. A paper is disqualified if it meets at least one of these exclusion criteria:

EC1. The domain of interest is not related with software testing or software performance, which means that a test is mentioned only to evaluate a system not related with software testing or software performance.
EC2. The research is not related with ontologies.
EC3. It is not written in English language.

The search string was designed to be highly generic in order to retrieve papers beyond the desired scope. However, the inclusion criteria was created aiming at filtering these additional results. Consequently, reviewers had an extra work to analyse a bigger amount of papers, but the risk of not retrieving relevant results was mitigated in this way. After applying those criteria, 199 articles from those 235 were removed, which represents 84.68% of the initial sample. The main reason that removed a paper is the lack of contribution to the domains of software testing or software performance (**EC1**). In other words, 36 articles (15.32% of the sample) met at least one of the inclusion criteria and did not meet any one of the exclusion criteria. It was found the whole text of only 33 articles from this list of 36 works, thus, 3 papers had to be removed from this review[2]. The 33 remaining articles were fully read, analysed and compared according with the research questions presented in sequence. During these stage 15 papers were removed from the review because of two different reasons: *(i)* the presented ontology-based approach was not clearly specified, or the ontology itself was missing, which removed 10 papers; and *(ii)* the paper was a shorter and older version of another paper already selected in this review, which removed 5 papers. Then, the resulting 18 researches ([1] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24]) were analysed and compared against the following questions.

[2]The papers removed from this review and the reasons can be found at https://code.google.com/p/performance-ontology/wiki/LiteratureReview

TABLE I. RESULTS OBTAINED WITH THE SEARCH STRING

Database	Raw Results	Excluded	Results
IEEExplore	137	5	132
Scopus	156	20	136
Compendex	57	11	46
Sum	350	36	314

How many works regarding ontologies in software testing or software performance domains propose new ontologies? What are the domains of these ontologies? New domain ontologies were proposed in 16 of these 18 papers, only [9] [19] used already developed ontologies. The domains addressed by each work are the following:

- Software testing, focusing on concepts such as test activities & artifacts [8]; test coverage criteria, test objectives & test cases [13] [16]; testers, test resources & activities [17] [18] [20]; test case, test process & test techniques [1]; test projects, test documents & test knowledge level [14]; test data, test result, test schedule & test plan [21]; and test case [12] [22].
- Software performance models, *i.e.*, workload, resource utilization, requests, servers and users [9] [10] [23] [24];
- Non-functional requirements, including concepts such as performance requirements & response time [11];
- Web services behaviour concepts [15] [19];

Which ontology-based applications in software testing or software performance domains are already developed? Application of ontologies are demonstrated in 15 of the 18 papers. This characteristic was missing at [8] [9] [11]. The application of ontologies for test case reused was explored in 3 papers: [1] [12] [22]. The performance assessment and improvement at runtime based on ontologies was investigated in 3 works: [10] [23] [24]. An ontology-based framework for test case generation is presented in [16] and an unit test case generation approach is explained in [13]. The generation of test case or test input data for web services, which usually includes automatic discovery and composition of web services, is investigated in 5 works: [15] [17] [19] [20] [21]. Finally, the use of a software test ontology for knowledge management was researched in [14] and his use to establish a reference architecture is presented in [18].

Which advantages were identified of using ontologies in software testing or software performance domains? Generally speaking, ontologies were used for knowledge representation, sharing, reuse, reasoning and inference. Specifically in the domains of software testing and software performance ontologies can be explored to enhance test case generation, foster test knowledge reuse, improve performance analysis and enable test tools interoperability. The advantages and contributions of ontology-based applications highlighted in the literature were diverse, as can be observed in the sequence. Representing data and reasoning about the environment state and the system performance in ontologies were advantages highlighted in [10]. According with [23], ontology reasoning allows the system to infer performance knowledge and take decisions about his own performance represented in ontologies. Similarly, in [24] it was studied the advantages of ontologies

to express performance-related information in a context-aware ambient intelligent application and the use of this knowledge to assess the application performance during its execution. In [11], ontology was used to support decision making in order to improve software performance and better achieve non-functional requirements. Test case specification and reuse by means of ontologies is proposed in [12]. Moreover, improving test case management is an advantage identified in [1] and ontology-based enhancing test case generation was researched in [16] [21]. In [21] the conclusion is that rules and reasoning techniques improve test generation intelligence and ontologies ease the sharing of testing assets. Semantic-level mutation testing analysis is another important contribution of using ontologies, as identified in [15]. In [17], it was concluded that ontologies enable automatic processing of testing tasks and standardize the vocabulary for encoding the information exchanged between testing services. According with [18], ontologies provide a mechanism to support the acquisition, organization, reuse and sharing of testing domain knowledge. Another interesting conclusion is that ontology reasoning can greatly improve the intelligence of automatic test generation, according with [19]. Testing process improvement, testing strategies definition and testing improvement measurements are advantages cited in [8]. Tool interoperability and model transformations were identified by [9]. In the realm of testing services, it was said that ontologies support dynamic discovery and invocation of testing services without compromising security, privacy and intellectual property rights [20].

Which limitations were identified of using ontologies in software testing or software performance domains? In [10], it was said that the proposed ontology-based approach to improve performance during execution can cause the opposite effect, which is performance degradation. This effect can result from a bad designed approach where the computation overhead used for reasoning exceeds the benefits obtained by the possible performance improvements. Moreover, the approach for ontology-based test generation presented in another work [13] can introduce the limitation of requiring knowledge engineering skills. Similarly, modelling complexities and time efficiency were cited as possible downsides of the test case generation approach proposed in [16]. In [17], it was concluded as limitation that is impossible to build a complete ontology of software testing given the huge volume of software testing knowledge and the rapid development of new testing techniques, methods and tools. As observed in [18], each ontology can cover certain knowledge aspects and use a different terminology. Finally, according with [24], there is no single correct ontology for any domain. Thus, ontology design is a creative process as modelling itself [24].

Which ontology technologies are more frequently applied? The ontology language most cited is OWL, in 14 of the 18 papers. The 4 works that did not cite any ontology languages or technologies were [9] [14] [16] [22]. Among the works that used OWL, the following 5 papers used an OWL extension known as OWL-S: [15] [17] [19] [21] [23]. The ontology tool most used is Protégé, as cited in the following 7 papers: [8] [12] [1] [17] [18] [19] [21]. Similarly, the Uschold methodology was the most cited approach for building ontologies, cited by the following 4 papers: [1] [10] [12] [23]. The most cited reasoner was Pellet, mentioned only by 2 works: [15] [21]. Another interesting technologies cited were Jastor [10] [23], POSL and OO jDREW [13], Jess [21] [23], SWRL [21], OWL API [17] [21], OWL-S API [17] [19] and Jena [15] [19] [21] [24].

Which sources are used to build the ontologies and which sources could be used to build ontologies? The main sources cited in the 18 papers were the following: SWEBOK [8] [1] [14]; Testing Maturity Model [8]; Capability Maturity Model [8]; UML SPT [9] [23]; Core Scenario Model and Software Performance Engineering Meta-Model [9]; RFC 2616 (HTTP/1.1) [10]; Non-Functional Requirement Framework Softgoal Interdependency Graph [11]; ISO/IEC 9126-1:2001 [1]; and UML Testing Profile [21]. Moreover, the following papers adapted and reused other ontologies, such as Time Ontology [23] [24]; Software Testing Ontologies [17] [20]; and Object Management Group Ontology Definition Meta Model [16]. On the other hand, the following papers did not explicitly cite any sources that were used or could be used to build ontologies [12] [13] [15] [18] [19] [22].

Which approaches are used to evaluate or compare ontologies in the domains of software testing or software performance? None of these 18 analysed papers addressed the issue of ontology validation. Also, comparisons among ontologies or even comparisons among papers that addressed ontologies in software testing or software performance domains were not found in the analysed literature. These characteristics indicates that this research topic is still in its infancy. Another related point is that none of these papers explicitly indicates a place where the ontologies or the ontology-based applications could be accessed and downloaded. However, it is possible that the ontologies and application proposed were not intended to be public, *e.g.*, they could be protected by some form of intellectual property. The unavailability of the ontologies complicates the overall understanding of the papers.

This section presented the results obtained from a literature review about ontologies on the domains of software testing and software performance. Although ontologies in related domains can be found (*e.g.*, a software testing domain ontology), this work is the first to propose an ontology on performance testing domain to answer the competency questions detailed in next section. We highlight that the trends and patterns identified in this literature review were used to guide decisions related with the ontology development approach adopted in this work.

III. Proposed Ontology

The motivation to build an ontology of performance testing lies in the assumption that the identification of tools, activities, metrics, artifacts and other domain concepts requires previous knowledge over the performance testing domain. Consequently, this research studies the possibilities of using ontologies to represent this knowledge and support tester's decisions. The ontology editor tool Protégé-OWL [3] was used to develop our ontology of software performance testing. The formalism that represents the ontology is OWL, which is a web standard language intended to explicitly represent the meaning of terms in vocabularies and the relationships between those terms [2]. In this context, it was adopted the ontology development methodology proposed in [25]. The first step of this methodology is to determine the ontology domain and scope. The ontology covers the domain of software performance testing and some examples of competency questions

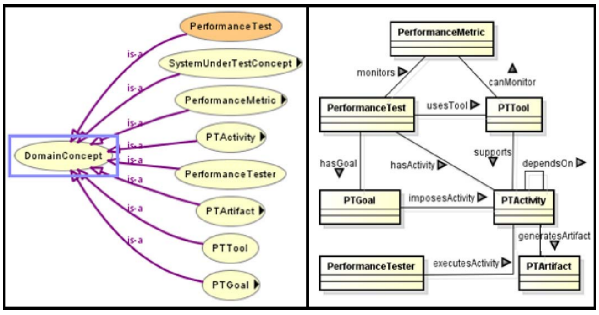

Fig. 1. Class hierarchy (left); main ontology concepts and properties (right)

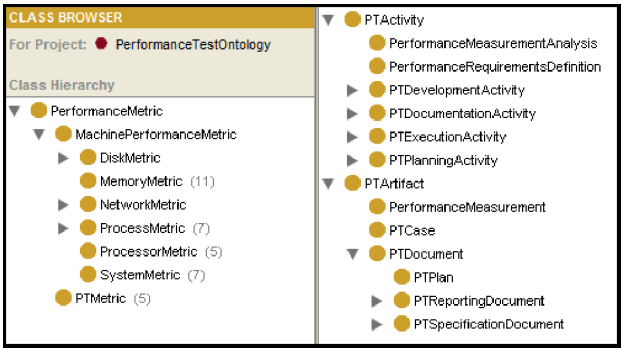

Fig. 2. Concept hierarchies of *PerformanceMetric*, *PTActivity* and *PTArtifact*

that it must answer are listed in the sequence. Which features should be considered during the preparation of a performance test? How to choose which technologies can be used according with the configuration of a system that will be submitted to a performance test? Which activities the performance test must have to achieve its goals? Which performance test tools can be used if the test objective is to monitor the Linux operating system? Given a test tool, which metrics it can collect and which protocols it is able to load test? The answers to these questions will be inferred from the concepts, properties and instances of our performance testing ontology. Constraints and capabilities involved in the application of each technology, *e.g.*, a capability provided by the performance testing tool *LoadRunner* is load testing applications over the HTTP protocol. Thus, once the test environment is defined, it is possible to choose technologies that can be used on a specific instance of performance test. Furthermore, according to the testers's choices in drafting a new performance test, a corresponding test plan can be generated as a result of the application that uses this ontology. The test plan is a document that provides the basis for the execution of software testing in a manner that is organized, systematic and well documented [26].

The second step of the methodology [25] is to consider the reuse of existing ontologies. Ontologies repositories were searched for OWL ontologies in related domains, since it was not found any ontology covering specifically the domain of performance testing. We analysed ontologies of software testing, such as OntoTest [27] and SwTO [28]. Our ontology was built based on concepts extracted from the Software Engineering Body of Knowledge (SWEBOK) [29], the IEEE Standard Glossary of Software Engineering Terminology [30] and the IEEE Standard for Software Test Documentation [26]. Then, the most important terms of the performance testing domain began to be defined as top classes and relationships, as shown in Figure 1. Some of these concepts and properties in the ontology use the acronym PT, which stands for *Performance Test*. Beyond the obvious performance test concept, the proposed ontology is composed of concepts such as performance test activities, tools, goals, artifacts (also known as deliverables), testers, performance metrics and concepts related with the system under test. Figure 1 shows the performance test concept (named *PerformanceTest*) with a color highlight. The reason is that *PerformanceTest* is an OWL defined class, while the other concepts are primitive classes. In OWL, primitive classes have restrictions that all individuals belonging to the class must satisfy, but it does not mean that a random individual that meets these conditions necessarily belongs to that class [2]. On the other hand, a defined class means that all individual belonging to that class must satisfy its class restrictions, and also means that any random individual that meets these class restrictions can be classified as belonging to that concept by semantic reasoners. According to this ontology, an instance of *PerformanceTest* must be related with at least one instance of *PTActivity* through the property *hasActivity*, as depicted in Figure 3. In addition, a performance test must have relationships with at least one application under test (*ApplicationUnderTest*), one goal (*PTGoal*), one tool (*PTTool*) and one performance metric (*PTMetric*) through the corresponding properties as defined at the *PerformanceTest* concept. Figure 1 shows a class diagram illustrating the main ontology concepts and relationships. The origin of a property denotes its domain, and its destination is the image (or range) of such relationship between concepts.

The test activities represent important concepts in this proposed performance testing ontology. Figure 2 points out the *PTActivity* concept hierarchy, while Figure 3 illustrates the axioms of two concepts that specialize *PTActivity*: *PTReportElaboration* and *PerformanceToolsDefinition*. Besides the *generatesArtifact* property, these concept have restrictions based on the *dependsOn* property, which is a transitive property having *PTActivity* concept both as domain and image. Because of the transitivity characteristic, if an activity A depends on an activity B and this activity B depends on an activity C, then it can be inferred that A depends on C. The activities of performance tests were classified in the ontology as development, documentation, implementation or planning activities. These categories are not mutually exclusive, for example, the activity responsible for creating the test plan (*PTPlanElaboration*) is classified both as documentation and planning activity. As Figure 1 illustrated, further properties of this concept explored in the ontology refer to the fact that an activity may be supported by a tool and also an activity can generate an artifact such as, for example, a test case.

Figure 2 also depicts the *PTArtifact* concept hierarchy, which was designed according with IEEE 829-1998 [26] taxonomy. In this standard, test documentation is divided into three types: planning, specification or reporting. These concepts can help the tester to determine which set of test documents will be used and what information each of these documents should include. Thus, the ontology provides a

standard description of test documents, which might serve as a reference to facilitate communication about the meaning of each test artifact. The performance metric is also an important concept of the ontology, which is divided into two main categories: the metrics related with machines and the metrics related with transactions. The machine metrics, represented by the *MachinePerformanceMetric* concept, are subdivided into more specialized categories, such as metrics of memory, processor, process and disk. This hierarchy is shown in Figure 2, which also indicates that these concepts already have instances. For example, the class *MemoryMetric* has as instances the amount of memory available in megabytes and the number of pages accessed by the second. On the other hand, the metrics related to a transaction are represented by the *PTMetric* class, which have instances to represent the response time and the throughput. Still regarding the metrics, the ontology has the property *canMonitor* to represent that a given test tool can monitor certain performance metrics. In other words, the property *canMonitor* has the class *PTTool* as domain and the class *PerformanceMetric* as range, as depicted at the right side of Figure 1. The *PTTool* concept was also already instantiate, it has 18 instances to represent performance test tools, such as, LoadRunner, Visual Studio, JMeter and so on. These instances were defined to represent the features offered by each tool.

The ontology defines 7 concepts as subclasses of *PTGoal* to represent different possible performance test goals. Figure 3 depicts the restrictions of one of these concepts, named *NormalWorkloadPerformanceEvaluation*. For example, if a test has the goal of determining the performance of a system (*NormalWorkloadPerformanceEvaluation*), then at least one instance of activity responsible for analysing the performance measurements (*PerformanceMeasurementAnalysis*) must be executed. This restriction is defined using the property *imposesActivity*, whose domain is *PTGoal* and image is *PTActivity*. As shown in Figure 3, goals are modelled as classes to allow the specification of constraints at concept level, and these classes only need a single instance to represent the test goal that will be used to specify performance test instances.

This section presented the proposed ontology for the domain of software performance test. Next section shows an approach to assess and compare the proposed ontology.

IV. Ontologies Comparison

Evaluating the quality of an ontology allows the identification of parts that can be better specified and enables the comparison between two or more ontologies. This section compares the proposed ontology with the two most related ontologies found in literature: OntoTest [27] and SwTO [28]. OntoTest [27] is an ontology of software testing built in order to support the acquisition, organization, sharing and reuse of knowledge regarding this domain. OntoTest explores the different aspects involved in the activity of software testing, defines a common vocabulary and also helps in the establishment of reference architectures. Such architectures simplify the use and integration of tools, processes and artifacts in software engineering environments. On the other hand, SwTO [28], which stands for Software Test Ontology, is an ontology concerning software testing designed for testing over the Linux domain. Hence, SwTO represents both software testing concepts and Linux operating system concepts [28], but without focusing on

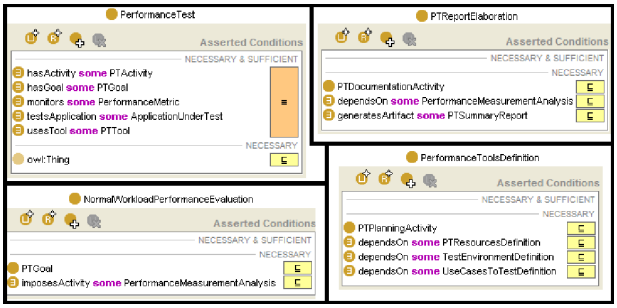

Fig. 3. Axioms on four different concepts: *PerformanceTest* (upper left), *NormalWorkloadPerformanceEvaluation* (lower left), *PTReportElaboration* (upper right) and *PerformanceToolsDefinition* (lower right)

performance testing. An ontology evaluation and comparison approach based on metrics describes and highlights certain aspects of each ontology, rather than classifying them as right-wrong, efficient-inefficient, and so on. From the viewpoint of ontology developers, metrics can be used to recognise areas that might need more work or cause problems. Table II shows metrics extracted in Protégé [3] for each ontology being compared. In principle, a bigger number concepts indicates that more domain elements were represented in the ontology. However, absolute metrics, such as the number of concepts, do not take into consideration whether these concepts are correct or if they will be useful. Moreover, Table II highlights that Ontotest [27] presents the highest number of concepts but the smallest number of relational properties. On the other hand, SwTO [28] was the ontology with more relational properties but less concepts. Also, the proposed ontology presents more predefined instances and axioms.

The performance testing domain has different concepts and properties when compared with these related ontologies, such as performance metrics, different testing goals and tools capabilities. On the other hand, upper concepts such as tester, artifact and activity are similar, although there are different subclasses of these concepts between the domains of software testing and performance testing. In other words, these related works address the software testing domain, but do not specify the performance testing sub domain. Table II illustrates the differences according to the number of subclasses among the main concepts represented in each of these ontologies. According to Table II, the proposed ontology specifies with more detail the concepts of test activity, test deliverable and performance metric. Although the SwTO represents 11 test goals, as opposed to the proposed ontology, not all these goals are applicable to the performance test domain. The differences among these ontologies are not only the numbers of concept subclasses shown in Table II. For example, the concept test tool exists in all ontologies, but in the proposed ontology it is used in 5 properties and it is related with 4 different concepts. The same concept is used in only 2 properties in the other ontologies. There are also divergences regarding the axioms that define each concept, properties and individuals represented in each ontology. These ontologies were designed based on distinct competency questions and domains. The focus of our ontology is the performance testing domain, thus it describes concepts not represented in other ontologies, such as the performance metrics, as shown in Table II.

TABLE II. ONTOLOGIES COMPARISON

Number of ...	PTOntology	Ontotest	SwTO
Concepts	99	**126**	85
Relational properties	44	19	**85**
Datatype properties	9	6	**26**
Instances	**171**	18	36
SubClassOf axioms	118	161	**166**
DisjointClasses axioms	**231**	0	180
AnnotationAssertion axioms	**187**	0	153
Test activity concepts	**29**	20	7
Test document concepts	**10**	8	4
Test goal concepts	7	0	**11**
Test deliverables concepts	**13**	7	0
Test tool concepts	**10**	0	1
Test tool instances	18	1	**32**
Performance metrics concepts	**23**	0	0

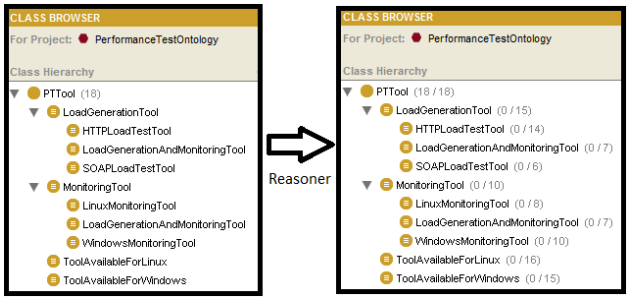

Fig. 4. Executing the semantic reasoner Pellet in the proposed ontology

V. ONTOLOGY-BASED APPLICATIONS

Applications using our ontology can be developed in different ways such as by extending the functionality of Protégé or using an API for ontology manipulation. The following subsections illustrate some of these ways of using the proposed ontology. Although these ontology-based applications employ different approaches, the goal is always to help the software tester to establish performance tests. The applications aim at supporting testers' decisions with domain knowledge of technologies used in performance testing software, validating and recommending options according with the test environment, goals and activities that tests might present. To achieve this, the applications must query the ontology that has knowledge about the activities required to conduct performance tests, the tools that can solve such activities, the requirements for using a given test tool and so forth. One assumption for such applications is that the identification of appropriate tools, activities, artifacts and metrics for a specific test requires some prior knowledge about the domain. Thus, applications might assist testers according with the knowledge represented in the ontology. To execute inferences over the domain, the available axioms in the ontology are manipulated to answer questions designed according with the ontology scope, which are as follows: According to the test environment, which tools can be used? According to the test goals, which activities should be performed? According with both the test goals and test environment, which performance metrics may be collected?

A. Using Semantic Reasoners

It is expected from an OWL semantic reasoner the functionalities of consistency checking, concept satisfiability, classification and realization [4]. Consistency checking ensures that the ontology does not contain contradictory facts; concept satisfiability checks if it is logically possible for a concept to have instances; classification computes hierarchical relationships between classes, and realization find concepts to which individuals belong [4]. In other words, semantic reasoners are able to infer logical consequences from an initial set of axioms. To use the reasoners feature known as realization, some concepts with restrictions were defined, for example, an individual is a *MonitoringTool* if it belongs to the *PTTool* concept, has the property *canCollectMetric* with an instance of *MachinePerformanceMetric* and also has the property *supports* with an instance of *PerformanceMetricMonitoring*. It is important to remember that more complex concepts can be build based on previous concepts, for example, the *LinuxMonitoringTool* concept is any individual that is a *MonitoringTool* and has at least one relationship with a *LinuxBasedOperatingSystem* individual through the property *canMonitor*. An example executing a semantic reasoner is presented in Figure 4, where 18 individuals were classified in the *PTTool* hierarchy using the Pellet [4] reasoner. In this example, from the 18 instances of *PTTool*, 15 were classified as *LoadGenerationTool*, 10 as *MonitoringTool*, and so on.

B. Executing Queries in SQWRL

Another way to explore the proposed ontology is using semantic queries represented in SQWRL (Semantic Query-enhanced Web Rule Language) [5]. SQWRL provides a simple and expressive language for querying OWL ontologies in a way that queries are serialized along with the ontology [5]. Below there is an example of a SQWRL query that retrieves HTTP (HyperText Transfer Protocol) load generation tools, which are instances of performance test tool that can generate load at the HTTP communication protocol: *PTTool(?x)* $\wedge$ *generatesLoadOn(?x, ?y)* $\wedge$ *sameAs(?y, HTTP)* $\Longrightarrow$ *sqwrl:select(?x, ?y)* $\wedge$ *sqwrl:columnNames("Performance Test Tool", "Communication Protocol")* $\wedge$ *sqwrl:orderBy(?y)*

C. Extending Protégé Software

Protégé can be configured to allow intuitively ontology visualization and manipulation by end users. Still, Protégé has an architecture that allows development of plugins that add new features to the tool [3]. Thus, it is possible to take advantage of the existing functionality of Protégé, for example, the use of semantic reasoners and execution of queries written in SQWRL. Protégé allows different kinds of plugins, in this work it was developed components known as tab widget and slot widget plugins. Tab widget plugins are new tabs that can be visualized at Protégé interface to implement new features. In contrast, a slot widget plugin is a graphical component that extends some Protégé graphical interface component aiming at expanding the viewing and editing information of ontology individuals and properties. The developed tab widget plugins were named *ActivitiesDependencies* and *TestPlanGenerator*, while the slot widget plugins received the names of *AnnotationsWidget* and *RestrictionsWidget*.

The *ActivitiesDepedencies* plugin is a new tab for Protégé which infers what activities should be part of a given performance test instance. To achieve this, the *hasActivity* property must be considered, whose domain is the *PerformanceTest*

concept and the image is *PTActivity*, indicating that a performance test might include multiple activities. In turn, each activity can present a dependency relationship with instances of *PTActivity* through the ontology property *dependsOn*. With this knowledge, a graph was generated, using the JUNG API[3] (Java Universal Network/Graph Framework), where the nodes are activities already included in a specific performance test instance plus the activities that should be added since they are directly or indirectly requirements of an included test activity.

The *TestPlanGenerator* plugin uses the iText API[4] to create a test plan in PDF containing the axioms related with a given instance of the *PerformanceTest* concept. To generate a test plan for a *PerformanceTest* individual, the algorithm starts with an *OWLIndividual* object (from the OWL API) and discovers all properties of the selected test using *getObjectPropertyValues* method. Then, for each property P, it writes that property characteristics in the PDF and the characteristics of each individual of the ontology referenced by that property P with the selected test. More details about the use of OWL API will be explained later in the paper.

By extending the class *SlotWidget* available in Protégé API (package edu.stanford.smi.protege.widget) it is possible to customize interface components in order to display features not available in the standard version. Graphical components extending *SlotWidget* are presented to the user when ontology concepts are instantiated. The *RestrictionsWidget* plugin allows the visualization of constraints for a given concept as represented in the ontology, which are the axioms imposing restrictions for individuals of a specific concept. On the other hand, the *AnnotationsWidget* component displays the annotations and comments of a particular ontology concept.

This approach of manipulating the ontology through Protégé offers the advantage of reusing already implemented features such as the instantiation of concepts and properties. The use of semantic reasoners (*e.g.*, for consistency checking) or semantic queries (*e.g.*, SQWRL) are other important features that can be highlighted. Moreover, the ontology parsing is transparently handled by the Protégé framework.

D. Developing an OWL API Application

Since our ontology is encoded in an OWL file generated by Protégé [3], it is possible to develop a custom application to manipulate its axioms. To investigate this approach, it was build a Java application that handles the axioms of this OWL file with the goal of enhancing performance tests planning and management. The OWL API [6] version 3.2.4 was used to access the knowledge in the ontology. This library is an open source Java API (Application Programming Interface) that enables the creation, manipulation, and serialization of OWL ontologies through classes such as *OWLOntologyManager*, *OWLOntology* and *OWLDataFactory*. The application aims to help the software tester to establish performance tests and generate corresponding test plans. To establish a performance test the user has to create an instance of the concept test performance in the ontology and link this instance with at least one goal, one application, one activity, one tool and one performance metric. Moreover, the application allows creating, retrieving and updating instances of the ontology concepts and properties, such as tools, activities, artifacts and performance metrics. The application enables the binding of a specific instance of performance testing with instances of test tools, activities, objectives, metrics and other concepts based on the properties in the ontology. The OWL API was used, for example, to create a new instance in the ontology and to add an assertion axiom that declares this new instance as belonging to a given concept. This is done in order to create new instances of concepts such as *PerformanceTest*, *PTActivity* and *PTTool*.

This section demonstrated different applications on top of the proposed ontology. Diverse technologies were investigated in practice during the development of these applications, such as, Protégé [3], Pellet [4], SQWRL [5], OWL API [6], Java, JUNG and iText API. Moreover, this section presented a number of technical details about the development of ontology-based applications for the domain of software performance testing. The following section concludes this paper.

VI. Final Remarks

This paper contributes with a literature review covering the use of ontologies in software testing and software performance. Its goal is to identify trends and patterns considering the state of art about ontology approaches in these domains. The results obtained in this review highlighted ontology applications already developed, advantages of using ontologies, ontology technologies most frequently applied, and so on. After this theoretical review, we investigated, in practice, the development of both a performance testing domain ontology and ontology-based applications built on top of it. The proposed software performance testing ontology was build with the best academic reported techniques, tools and methodologies from the literature of ontological engineering. The ontology was represented in OWL [2] language, developed inside Protégé tool [3], according with the methodology reported in [25]. The concepts, relationships, properties, axioms and instances that represent the knowledge of the ontology were extracted from the literature of the domain such as the IEEE Standard Glossary of Software Engineering Terminology [30], the IEEE Standard for Software Test Documentation [26] and the Software Engineering Body of Knowledge (SWEBOK) [29]. In this context, OWL ontologies of software testing, such as OntoTest [27] and SwTO [28], were also studied, and our ontology was evaluated by comparing it with these related ontologies through metric-based approaches.

The proposed ontology was explored by means of semantic reasoners (*e.g.*, Pellet [4]), semantic queries (in SQWRL [5]), custom developed Protégé plugins [3] and Java applications using OWL API [6]. This ontology was used as the basis for applications designed for planning and management of performance tests. Since ontologies are representations of domain knowledge that enable knowledge sharing between different applications, this work aimed at investigating their practical potential. Although other ontologies in related domains have been proposed, this work is the first to tackle the performance test domain, including specific concepts and properties not represented in the software testing field. Moreover, the ontology can be extended with new concepts, such as *EnduranceTest* or *StressTest*, which would be both subclasses of *PerformanceTest*, but the former would include a restriction

[3]JUNG API is available open source at http://jung.sourceforge.net/

[4]iText API is available at http://itextpdf.com/

upon its instances of including at least one relationship with an instance of *LongRunPerformanceEvaluation*, and the later could be defined as requiring a relationship with a *PerformanceBottleneckIdentification*. Similarly, other concepts could be specialized, for example, *PerformanceTester* with concepts such as *TestAnalyst*, *TestDeveloper*, *TestSenior*, and so on. As advantages of using ontologies, we can mention that a software performance ontology standardizes the vocabulary of the area and facilitates the exchange of knowledge [27]. Besides, ontologies can improve learning about the domain since they provide a shared understanding of the concepts and relationships on the field [27]. Other usage examples of ontologies identified in related papers are for test case generation [13] and runtime performance improvement [10]. We investigated if ontology, as a knowledge representation technique, can be applied in the performance testing process, for example, to make better decisions, decrease costs and increase quality. For this reason, the proposed ontology-based applications have the goals of supporting technical decisions over performance testing domain and enabling access both to domain knowledge and knowledge of previous tests. Thus, it can be concluded that, since software testing requires skilled testers with both technical as well as domain knowledge, there is a gap that ontology approaches can fulfil[5].

REFERENCES

[1] L. Cai, W. Tong, Z. Liu, and J. Zhang, "Test Case Reuse Based on Ontology," in *2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*, 2009, pp. 103–108.

[2] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "OWL Web Ontology Language Reference," W3C, http://www.w3.org/TR/owl-ref/, Tech. Rep., February 2004.

[3] Stanford Medical Informatics, "Protégé software web site," 2005. [Online]. Available: http://protege.stanford.edu/

[4] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Journal of Web Semantics*, vol. 5, no. 2, pp. 51–53, Jun. 2007.

[5] M. J. O'Connor and A. K. Das, "SQWRL: A query language for OWL," in *OWLED*, ser. CEUR Workshop Proceedings, R. Hoekstra and P. F. Patel-Schneider, Eds., vol. 529. CEUR-WS.org, 2009.

[6] M. Horridge and S. Bechhofer, "The OWL API: A Java API for OWL ontologies," *Semantic web journal*, vol. 2, no. 1, pp. 11–21, Jan. 2011.

[7] B. Kitchenham, R. Pretorius, D. Budgen, O. Pearl Brereton, M. Turner, M. Niazi, and S. Linkman, "Systematic literature reviews in software engineering - a tertiary study," *Inf. Softw. Technol.*, vol. 52, no. 8, pp. 792–805, Aug. 2010.

[8] H. Ryu, D.-K. Ryu, and J. Baik, "A strategic test process improvement approach using an ontological description for MND-TMM," in *Proceedings of the Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*, ser. ICIS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 561–566.

[9] V. Cortellessa, "How far are we from the definition of a common software performance ontology?" in *Proceedings of the 5th international workshop on Software and performance*. ACM, 2005, pp. 195–204.

[10] C. Guerrero, C. Juiz, and R. Puigjaner, "Web Performance and Behavior Ontology," in *International Conference on Complex, Intelligent and Software Intensive Systems*. IEEE, 2008, pp. 219–225.

[11] P. P. Sancho, C. Juiz, R. Puigjaner, L. Chung, and N. Subramanian, "An approach to ontology-aided performance engineering through nfr framework," in *Proceedings of the 6th international workshop on Software and performance*. ACM, 2007, pp. 125–128.

[12] S. Guo, J. Zhang, W. Tong, and Z. Liu, "An application of ontology to test case reuse," in *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*. IEEE, Aug. 2011, pp. 775–778.

[13] V. H. Nasser, W. Du, and D. MacIsaac, "Knowledge-based software test generation," in *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2009)*, 2009, pp. 312–317.

[14] L. Xue-Mei, G. Guochang, L. Yong-Po, and W. Ji, "Research and implementation of knowledge management methods in software testing process," in *2009 WRI World Congress on Computer Science and Information Engineering*, 2009, pp. 739–743.

[15] S. Lee, X. Bai, and Y. Chen, "Automatic Mutation Testing and Simulation on OWL-S Specified Web Services," in *41st Annual Simulation Symposium (anss-41 2008)*, Apr. 2008, pp. 149–156.

[16] V. H. Nasser, W. Du, and D. MacIsaac, "An ontology-based software test generation framework," in *Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering*, 2010, pp. 192–197.

[17] H. Zhu and Y. Zhang, "Collaborative testing of web services," *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 116–130, 2012.

[18] E. Y. Nakagawa, E. F. Barbosa, and J. C. Maldonado, "Exploring ontologies to support the establishment of reference architectures: An example on software testing," in *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, Sep. 2009, pp. 249–252.

[19] Y. Wang, X. Bai, J. Li, and R. Huang, "Ontology-Based Test Case Generation for Testing Web Services," in *8th International Symposium on Autonomous Decentralized Systems*, Mar. 2007, pp. 43–50.

[20] Y. Zhang and H. Zhu, "Ontology for Service Oriented Testing of Web Services," in *2008 IEEE International Symposium on Service-Oriented System Engineering*, Dec. 2008, pp. 129–134.

[21] X. Bai, S. Lee, W.-T. Tsai, and Y. Chen, "Ontology-Based Test Modeling and Partition Testing of Web Services," in *2008 IEEE International Conference on Web Services*. IEEE, sep 2008, pp. 465–472.

[22] X. Li and W. Zhang, "Ontology-Based Testing Platform for Reusing," in *2012 Sixth International Conference on Internet Computing for Science and Engineering*, Apr. 2012, pp. 86–89.

[23] I. Lera, P. P. Sancho, C. Juiz, R. Puigjaner, J. Zottl, and G. Haring, "Performance assessment of intelligent distributed systems through software performance ontology engineering (SPOE)," *Software Quality Journal*, vol. 15, no. 1, pp. 53–67, Jan. 2007.

[24] I. Lera, C. Juiz, R. Puigjaner, C. Kurz, G. Haring, and J. Zottl, "Performance assessment on ambient intelligent applications through ontologies," in *Proceedings of the 5th international workshop on Software and performance*. ACM Press, 2005, pp. 205–216.

[25] N. F. Noy and D. L. mcguinness, "Ontology Development 101: A Guide to Creating Your First Ontology," Online, 2001. [Online]. Available: http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html

[26] IEEE, "IEEE 829-1998 – Standard for Software Test Documentation," Tech. Rep., 1998. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=741968

[27] E. F. Barbosa, E. Y. Nakagawa, and J. C. Maldonado, "Towards the establishment of an ontology of software testing," in *18th International Conference on Software Engineering and Knowledge Engineering (SEKE 2006)*, San Francisco, CA, July 2006, pp. 522–525, short Paper.

[28] D. Bezerra, A. Costa, and K. Okada, "SwTO (Software Test Ontology Integrated) and its application in linux test," in *International Workshop on Ontology, Conceptualization and Epistemology for Information Systems, Software Engineering and Service Science*, 2009, pp. 25–36.

[29] A. Abran, J. W. Moore, P. Bourque, R. Dupuis, and L. L. Tripp, *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. IEEE, 2004, ISO Technical Report ISO/IEC TR 19759. [Online]. Available: http://www.swebok.org/

[30] IEEE, "IEEE Std 610.12-1990 – IEEE standard glossary of software engineering terminology," Tech. Rep., Dec. 1990. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=159342

[5]Ontology available at https://code.google.com/p/performance-ontology/