SwTO^I (Software Test Ontology Integrated) and its Application in Linux Test

Daniella Bezerra[1], Afonso Costa[2] and Karla Okada[2]

[1] Paulo Feitoza Foundation - FPF, Manaus - Brazil
daniella.bezerra@fpf.br
[2] Nokia Institute of Technology - INdT, Manaus - Brazil
afonso.costa@indt.org.br
karla.gomes@indt.org.br

Abstract. This paper encompasses a elements study of knowledge representation founded on ontologies that have Linux test as target domain. The study aims demonstrating that once knowledge is formalised, it is possible to reuse, to perform inference, to process it through computers, and, what is more relevant, it becames enable to be communicated between people and software. Towards that, three ontologies have been developed: OSOnto (Operating System Ontology) which represents concepts of the operating system domain, SwTO (Software Test Ontology) which deals with the software testing domain, and SwTO^I (SwTO Integrated) which represents concepts of both domains in an integrated way. For implementing the ontologies, OWL DL as ontology specification language, Protégé as ontology editor and Racer as the main reasoner, have been used. A quantitative and qualitative evaluation of the SwTO^I ontology has been performed as well.

Key words: knowledge representation, ontologies, linux test, evaluation

1 Introduction

One of the most challenges that the Software Engineering faces is: who does build software in a good quality with the reduced cost, time and effort? Several technological solutions are aimed as answers for this asks. One of them is the Free Software that seeks to contribute with its collaborative projects substantiated in the knowledge liberty and the intellectual property. Among the several free software projects, the Linux[3] is detached to have been one of the pioneers and for grouping a significant number of collaborators and users. The project maturity and the technical group, formed by people that participate actively, are interesting points to be observed. However, the quality of the Linux is much more observed and felt by users than actually verified by scientific procedures. Linux has been inserted in the business domain, then its quality needs to be actually verified. This fact results in a union between communities and interested

[3] http://www.linux.org

companies, giving rise to the Linux Test Project (LTP)[4]. The objective of this project is assurance the Linux quality through systematic tests. This project behaves as a repository with scripts, tools and more of 3.000 test cases. The LTP documentation is deficient and there is not an efficient record of the designers knowledge in the tests elaborations. These difficulties inside the Linux test community is common in many others projects and this has motivated the research and it has awakened the interest to offer an ontological contribution based in Description Logic, at least reducing these difficulties of the Linux test knowledge, which is maintained and recorded in a formality scarce approach, such as e-mail lists, source-code documentation and web forums. Towards that, three ontologies have been developed: OSOnto (Operating System Ontology) which represents concepts of the operating system domain, SwTO (Software Test Ontology) which deals with the software testing domain, and SwTO^I (SwTO Integrated) which represents concepts of both the above domains in an integrated way. For implementing the ontologies, OWL DL as ontology specification language, Protégé as ontology editor and Racer as the main reasoner, have been used. SwTO^I is used by TeSG (Test Sequence Generator) Tool to create some tests for LTP. An evaluation of the SwTO^I has been performed and some results has been included in this paper.

2 Related Domain Ontologies

A research of existing ontologies for the identified domains was carried out with the objective to reuse knowledge. For the operating system domain, the more significant ontology found was BLO (*Basic Linux Ontology*)[5], which includes concepts of a Linux introductory course of the Leeds University [3]. Already in the software test domain, the ontology more significant was SWEBOK (*Software Engineering Body of Knowledge*) proposal by a big technical IEEE committee, search supply definitions and terminology for the several knowledge areas of the software engineering and among them is the software test [7]. From the formal ontology BLO, it was possible to extract and reuse knowledge in the form of classes, properties and instances. From the informal ontology SWEBOK, it was possible to extract the knowledge of the text in natural language and transcribe it for the OWL DL language, giving rise to a formal ontology, SwTO. The activity of capturing concepts includes the identification, description and choice of concepts names associated to operating system domain and software test, as well as the identification of the relationships among them.

3 LTP and the selected scope

Linux has several subsystems, like network, memory, file system, etc. The research was delimited in a deliberate sample of the Virtual File System (VFS)

[4] http://www.ltp.sourceforge.net/

[5] *Basic Linux Ontology* was built by a specialist of the domain utilizing the language OWL-Lite and the Protégé. This ontology is part of a bigger project called SWALE.

because the LTP contains several test cases for those subsystems and the VFS is strategic for the Linux kernel[6]. Nevertheless, the VFS has four objects that can be investigated (inode, superblock, dentry and file) among the which we have selected file. The initial analysis of the Linux test domain was deed through bibliographical researches and meeting with specialists. Through the elaboration of a diagram of the conceptual model was possible to visualize the presence of two distinct domains, software test and the operating system. By a decision of project, we opted by developing an ontology for each one of the identified domains, permitting in this way modulate them. The OSOnto (Operating System Ontology) concentrates in the domain of operating system, the SwTO (Software Testing Ontology) concentrates in the software test domain. However, those domains are able to be integrated, what gave rise to the SwTO^I (Software Test Ontology Integrated)[7]. In this way, the knowledge about the Linux test forms a possible one ABox for the SwTO^I, which is discussed in the following section.

4 SwTO^I Kernel used by TeSG to generate test sequences

The software test method proposed in [1] contemplates TeSG (Test Sequence Generator) and offers a solution that identifies critical fails in the software. For so much, TeSG uses the knowledge represented by the SwTO^I. But why an ontology is used by TeSG? Why SwTO^I was chosen?

The use cases are textual documents, described in natural language. For they can be interpreted and processed computationaly, it is necessary to describe them formally. Transcribing use cases of a natural language for a computational language is not sufficient. Generating test sequences with probability to find errors or software bugs, it is necessary to unite the designer knowledge about the software test theory and associate it with test cases elaboration. This union between knowledge representation and its instanciation can be obtained through formal ontologies. This benefit did with TeSG considered ontologies as an adequate formality to be utilized in the problem solution.

The research carried out in [1] aims SwTO^I as the formal ontology more prominent for the TeSG application domain, for this, SwTO^I was chosen. But how this ontology is used by TeSG?

The *first stage* consists in representing narrative uses cases to formal use cases as instances of SwTO^I. In this case we have part of *UseCase* formalization illustrated subsequently in Figures 1 and 2:

UseCase: it is a documentation, which each use case has one or more settings to define how the system should interact with its actors for attending an objective or a task realization.

Sufficient and Necessary Condition:

(i) Individuals are inferred like a UseCase instances if they will be instances of Expanded class or HighLevel class since the subclasses are disjoin.

$$\texttt{UseCase} \equiv \texttt{Expanded} \sqcup \texttt{HighLevel} \quad (1)$$

[6] http://www.kernel.org

[7] A complete specification about the three ontologies are available in [2]

Among several Necessary Condition of this class we have:

(i) **UseCase** is subclass of **DesignDocumentation**.

$$\texttt{UseCase} \sqsubseteq \texttt{DesignDocumentation} \tag{2}$$

(ii) Individuals of the *UseCase* class relate itself through *hasActor* property with individuals of the *Actor* class.

$$\texttt{UseCase} \sqsubseteq \forall\, \texttt{hasActor.Actor} \tag{3}$$

(iii) Individuals of the *UseCase* class relate itself through *hasActor* property with at least one individual of the class *Actor*.

$$\texttt{UseCase} \sqsubseteq \exists\, \texttt{hasActor.Actor} \tag{4}$$

Properties:

(i) **hasActor** is an inverse object property of the **isActorOf**, whose domain is **UseCase** and range is **Actor**.

$$\texttt{hasActor} \in \texttt{P}_0 \tag{5}$$

$$\texttt{hasActor} \equiv \texttt{isActorOf}^{-} \top \sqsubseteq \forall\, \texttt{hasActor} \,.\, \texttt{UseCase} \tag{6}$$

$$\top \sqsubseteq \forall \begin{pmatrix} \texttt{hasActor}^{-} \,. \\ \texttt{Actor} \end{pmatrix} \tag{7}$$

(ii) **hasRanking** is a functional object property, whose domain is **UseCase** and range is **ImportanceRanking**.

$$\texttt{hasRanking} \in \texttt{P}_0 \tag{8}$$

$$\top \sqsubseteq (\leq\ 1\ \texttt{hasRanking}) \tag{9}$$

$$\top \sqsubseteq \forall\, \texttt{hasRanking} \,.\, \texttt{UseCase} \tag{10}$$

$$\top \sqsubseteq \forall \begin{pmatrix} \texttt{hasRanking}^{-} \,. \\ \texttt{ImportanceRanking} \end{pmatrix} \tag{11}$$

(iii) A use case has priority. The functional data type property **hasUseCasePriority** represents this priority (Equation 12), whose domain is **UseCase** (Equation 14) and range is **XMLSchema:int** (Equation 15).

$$\texttt{hasUseCasePriority} \in \texttt{P}_\texttt{D} \tag{12}$$

$$\top \sqsubseteq (\leq\ 1\ \texttt{hasUseCasePriority}) \tag{13}$$

$$\top \sqsubseteq \forall\, \texttt{hasUseCasePriority} \,.\, \texttt{UseCase}\ (\texttt{UseCase} \neq \bot) \tag{14}$$

$$\top \sqsubseteq \forall \begin{pmatrix} \texttt{hasUseCasePriority}^{-} \,.\, \texttt{XMLSchema:int} \\ (\texttt{XMLSchema:int} \neq \bot) \end{pmatrix} \tag{15}$$

This property has integer values defined by owl:oneOf constructor. It can have priority from 1 until 5, where 1 means lower priority and 5 high priority.

$$\{\ 1, 2, 3, 4, 5\ \} \sqsubseteq \texttt{hasUseCasePriority} \tag{16}$$

Disjoin:

(i) **UseCase** class is disjoin of the **EventSequence**, **Actor**, **ImportanceRanking**, **ExternalBehavior**, **SoftwareRequirement**, **Event** and **Frontier**. These classes do not share instances.

$$\texttt{Event} \sqsubseteq \begin{pmatrix} \neg\ \texttt{EventSequence}, \\ \neg\ \texttt{Actor}, \\ \neg\ \texttt{ImportanceRanking}, \\ \neg\ \texttt{ExternalBehavior}, \\ \neg\ \texttt{SoftwareRequirement}, \\ \neg\ \texttt{Event}, \\ \neg\ \texttt{Frontier}, \end{pmatrix}$$

Instances of the *Use Case*:

$$\begin{gathered} \texttt{isGeneratedBy}(\texttt{ftest01}, \texttt{IBM\_TestCaseGenerationProcess}) \\ \texttt{isTestDocumentationOf}(\texttt{ftest01}, \texttt{LinuxTestProcess}) \\ \texttt{hasInputSpecification}(\texttt{ftest01}, \texttt{lseek}) \\ \texttt{hasInputSpecification}(\texttt{ftest01}, \texttt{read}) \\ \texttt{hasInputSpecification}(\texttt{ftest01}, \texttt{write}) \\ \texttt{hasInputSpecification}(\texttt{ftest01}, \texttt{truncate}) \\ \texttt{hasInputSpecification}(\texttt{ftest01}, \texttt{ftruncate}) \\ \texttt{hasInputSpecification}(\texttt{ftest01}, \texttt{fsync}) \\ \texttt{hasInputSpecification}(\texttt{ftest01}, \texttt{sync}) \\ \texttt{hasInputSpecification}(\texttt{ftest01}, \texttt{fstat}) \\ \texttt{hasOutputSpecification}(\texttt{ftest01}, \text{flag 0 for unexpected exit}) \\ \texttt{hasPurpose}(\texttt{ftest01}, \begin{pmatrix} \text{Testing file input} \\ \text{output operations} \end{pmatrix}) \\ \texttt{hasProcedure}(\texttt{ftest01}, \text{read}) \\ \texttt{hasProcedure}(\texttt{ftest01}, \text{write}) \\ \texttt{hasProcedure}(\texttt{ftest01}, \text{remove}) \\ \texttt{canBeMappedInto}(\texttt{ftest01}, \texttt{ReadFile}) \\ \texttt{canBeMappedInto}(\texttt{ftest01}, \texttt{WriteFile}) \\ \texttt{canBeMappedInto}(\texttt{ftest01}, \texttt{RemoveFile}) \end{gathered}$$

The *second stage* consists in, using Jena[8] Framework, manipulate the source code of SwTO^I with all specifications and instances to generate a schema. The TeSG uses this schema to manipulate TBox and ABox of SwTO^I.

Developed in Java, TeSG uses a genetic algorithm as an heuristic approach to combine events between related use cases. SwTO^I concentrates this knowledge about "Who has relation with" and "What is the use case priority". The Figure 1, extracted from Protégé, presents logical conditions of *UseCase* class.

The Figure 2 presents the properties that describes *UseCase* class. For this description is possible to observe:

– The properties *hasGoal, hasPostCondition, hasUseCaseIdNumber, hasUseCaseName, hasUseCasePriority* and *hasDocumentationVersion* are datatype properties.

[8] http://jena.sourceforge.net

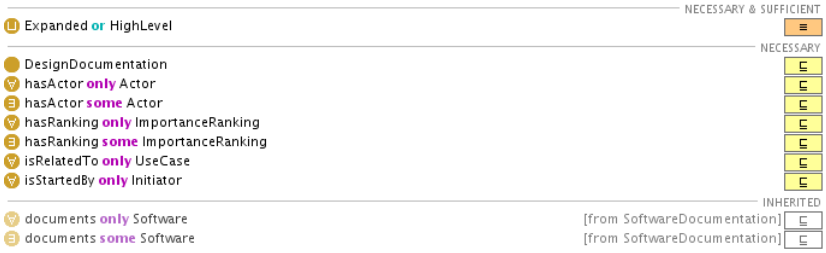

Fig. 1. Logical Description of the *UseCase* class

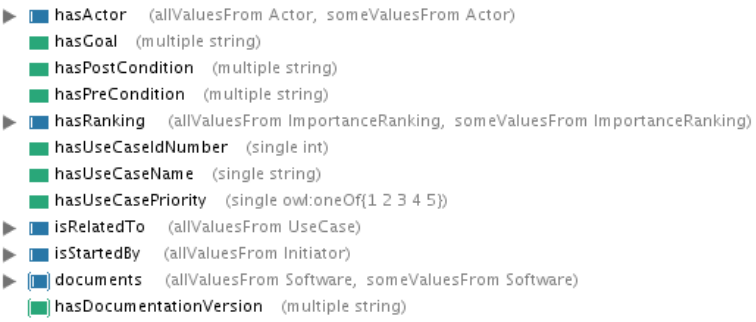

Fig. 2. *UseCase* properties

- The properties *hasActor, hasRanking, isRelatedTo, isStartedBy* and *documents* are object properties.
- The properties *hasPostCondition* and *hasPreCondition* are very important for test sequence generation. TeSG evaluates if a test case has pre or post conditions. If it has, this knowledge will compose a test sequence.
- The domain especialist is the person who defines the priority of use case through *hasUseCasePriority* property where 5 is the maximum priority and 1 is the minimum priority.

The *third stage* is generating a test sequence according to [1]. After this, the test sequence contemplates steps that can be executed like a black box test or can be implemented in an automated test case language.

5 SwTO^I Evaluation

The evaluation's purpose aims the quality of an ontology. Some works concentrate in the development and approaches studies, for so much, there are some proposed tools to support this activity, but they are not much utilized [6].

The SwTO^I evaluation was based on two strategies in parallel: quantitative and qualitative evaluations.

The quantitative evaluation was based on the approach described in [5], and it was also enriched with new criteria as class of bigger rank and levels of the

ontology. The qualitative evaluation was based on the approach described in [8], and it was enriched with the racer and metric utilization. Both strategies are complete and they are described in the next sections.

5.1 Quantitative Evaluation

The ontologies are used to represent knowledge of the most varied domains. They can present components in common as concepts, instances, attributes (of concepts or instances), relations that can be hierarchical (all-part and is-a) or not and restraints about concepts and relations in the form of axioms. The analysis of those components of an ontology can reveal information and important characteristics , such as, the level of ontology formality according to the explicit specification. Informal ontologies contemplates concepts, instances, relations and attributes. Formal ontologies, beyond contemplate the same components of the informal ontologies, also represent restrictions about concepts and relations in the form of axioms.

The approach proposed in [5] suggests an evaluation based on the ontology structure. This approach presents the ontology's components quantitatively and uses the graph's theory and statistical indicators. The approach also seeks to aim the level of ontology structure. An ontology well organized can easily supply the comprension of the represented domain and this is a good indicator of applicability, integrability and reusability. The approach also makes feasible the analysis and comparison among ontologies even though they represent distinct domains fairly by concentrate in ontological components.

In OWL DL a property is a binary relation. In this work we have considered two kinds of properties: object properties(P_O) and datatype properties(P_D). The quantitative evaluation was based on five indicators: (i) quantity of classes nominated, (ii) Medium of the properties P_O, (iii) is-a relation levels of the Ontology, (iv) class bigger rank of the is-a relation and (v) class bigger rank of the all-part relation. The isolated analysis of each one of those indicators does not lead to conclusive results. It is necessary to observe them together. The indicators will be detailed bellow.

Quantity of Nominated Classes - The quantity of nominated classes can supply an indicator of the ontology size and domain represented. SwTO^I has 194 nominated classes. However, this indicator is not conclusive. To evaluate the detail level of an ontology it is necessary other indicators. **Average of the Properties** P_O - An object properties reflects a relation between instances of two classes. The reasoners carry out inferences about object properties like the verification of restrictions about properties and the verification in domain/range axiom. A datatype properties reflects a relation between an instance and a XMLSchema type. The datatype properties are treated separately by the reasoners. Generally, ther is an inference machine separated for kinds of datatypes. Due to "open world assumption" logical premise of the reasoners utilized in OWL, the average analysis of the object properties regarding the total of nominated classes gives an idea of the representative distribution among the instances of the classes. SwTO^I has 142 P_O and 51 P_D.

In the following equation, MPO means the average of the ontology object properties; P_O is the total object properties; n is the total of ontology nominated classes.

$$MPO = \frac{P_O}{n}$$

The average of SwTO^I in relation of the nominated classes total is 0,73. This indicator can help the ontology engineers evaluates the abundance of relations among classes. **Is-a Relation Levels of the Ontology** - Upon observing the ontology structure, it is possible do an analogy with the graph's theory where a class corresponds to a node and the relation between classes (is-a or all-part) corresponds to an edge. The graphs are rich in properties and many information can be extracted from its analysis. From only one ontology, several graphs can be generated. If we consider the relation is-a between the classes, we have a tree and if an ontology imports another one, we have a forest. From this kind of graphs, it is possible to identify the ontology depth level, starting from the concept of the highest level or root concept until arrive the leaf concepts.

The Figure 3 generated by Jambalaya plug-in illustrates the classes tree and relations is-a of the SwTO^I. By this tree, it is possible to observe that this ontology has seven levels since the root to the leafs being that the fifth level of the root for the leafs has more density, with bigger quantity of classes. This in-

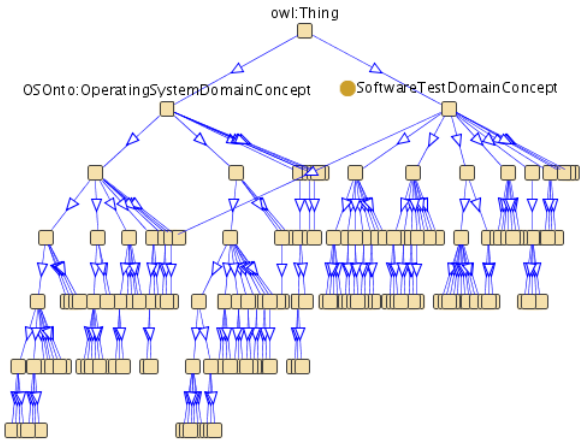

Fig. 3. SwTO^I tree and relation is-a between classes

dicator contributed with the ontology size evaluation, revealing the proportions of the root/leafs and the ontology depth through its hierarquical structure. The table 1 presents the total of classes by level and detaches the fifth one as the common level that concentrates the biggest quantity of classes. **Class Rank of the is-a Relation** - From the graph discussed in the previous section, it is possible to extract the class rank of the ontology. This indicator can reveal the

Level	SwTO^I
1	1
2	2
3	17
4	48
5	**70**
6	38
7	18
Classes Total	194

Table 1. SwTO^I Classes Total

classes that possess bigger connection (IS-A) with other and contribute to the project decisions as much as for the SwTO^I improvement and for other application that have the intention of reuse it. SwTO^I has *SoftwareTest* as a class with 11 IS-A relation. The SwTO^I's domain is the software test, then it is expected that an important class for domain has the greatest number of relations is-a. **Class Rank of the all-part Relation** - The relation all-part also establishes a kind of hierarch among the individuals of a class. As an example for this kind of relation, suppose that a *car* is instance of the class *vehicle* and the car is composed by the following parts: motor, chassis, etc. This kind of relation enrich a representation sets out of a domain informing the part of an all. For elaborating a graph of the relation all-part, it is important to consider the characteristics assumed by the properties like inverse, for example. SwTO^I has *SoftwareTestProcess* as a class with 20 ALL-PART relation. The Table 2 summarizes the indicator discussed in this Section and presents the results found for the SwTO^I. The SwTO^I's expressivity was identified by Protégé Metrics

Indicators	SwTO^I
Total of the nominated classes	194
P_O Total	142
P_D Total	51
P_O Average	0,73
Levels (is-a relation)	7
Class Rank (is-a relation)	*SoftwareTest*(11)
Class Rank (all-part relation)	*SoftwareTestProcess*(20)

Table 2. Results of the quantitative evaluation

as $\mathcal{SHOIQ}(D)$. This means that ontology contemplate transitive rules, intersection between classes, negation (complement of a class), full universal and existential quantification, and disjunction between classes. Also there is hierarch of properties, value restriction, inverse properties, symmetrical properties, well like cardinality restrictions.

5.2 Qualitative Evaluation

The qualitative evaluation was based on three criterias: consistency, completeness and conciseness.

In the purpose of an ontology, a determined definition is consistent if there is not contradictory knowledge inferred from axioms. The *consistency criteria* aims results of inferences carried out by a racer about the theory represented by the ontology.

If all of the concepts of a determined domain will be represented by an ontology and such representation will be verified by a test of completeness, this ontology is dictated complete. An ontology is an explicit specification of a conceptualization, that is going to offer an abstract and simplified vision of the world that it desires represent for some purpose. Represent all concepts can become impracticable for certain domains. The *completeness criteria* evaluates the explicit representation of the concepts and it aims an analysis of the ontology completeness.

An ontology is considered concise when: (i) does not store unnecessary definitions, (ii) does not exist redundancy between terms definitions , (iii) redundancies are not inferred of other definitions. The *conciseness criteria* evaluates each one of those points and aims an analysis of the conciseness of the ontology.

RacerPro supports the consistency and conciseness criteria only. Inferences were carried out in the ontology through the interface DIG (*Description Logic Implementers Group*), which works like a protocol of communication, allowing the reasoner interacts with the environment Protégé and consequently, with the source code of the ontology. **Consistency Criteria** - The reasoners help the consistency verification in three forms: (i) verifying some definite condition results in contradictory conclusions, (ii) inferring hierarchy of classes and subclasses and (iii) computing the instances inferred. The Protégé, used in partnership with a reasoner also helps this verification showing the hierarchy defined by the ontology engineer (asserted hierarchy) and after execution of the reasoner, the Protégé shows inferred hierarchy. If some class was reclassified, i. e., if superclass changes, the editor will detach in blue the hierarchy inferred. If the class is unstable, will be detached in red. The computation of the inferred instances, carried out by the reasoner, also can be visualized by the Protégé, which shows the instances associated to each class.

The RacerPro concluded that SwTO^I is consistent since no definite condition resulted in contradictory conclusions. This same consistency verification was carried out for the OSOnto and SwTO. Both also they are consistent. Right away was carried out the verification of the taxonomia.

The class *Agent* is subclass of the union between *OSOnto:Person* and *OSOnto:Software*. This specification results in a reclassification of class *Agent* like subclass of *OSOnto:OperatingSystemDomainConcept*. The class *GrayBox* is a software test technique equivalent the union of other two test techniques disjoin among themselves, to *BlackBox* and to *WhiteBox*. In the definite hierarchy those three classes are sisters, but in the inferred hierarchy, *BlackBox* and *WhiteBox* are subclasses of textitGrayBox. The class *GrayBox* is equivalent to

a cover axiom, defined that all instances belong also to one of its subclasses, i. e., all technical *GrayBox* will be exclusively *BlackBox* or *WhiteBox*. The Figure 4 presents an alert of the Protégé for the inferred reclassification. The Racer

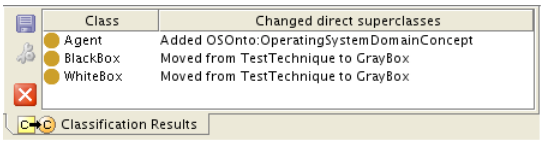

Fig. 4. Protégé result for SwTO^I classification

also contributed with the instances classification of each class. All the presented instances in the ontology were classified correctly. **Completeness Criteria** - The classes can be classified as primitive or defined. A primitive or partial class is one which has only necessary conditions. The defined or complete classes are those which have at least a sufficient and necessary condition. All class that is not defined as primitive since all class will have at least a necessary condition (all class will be subclass of some class, neither that be of owl:Thing). Evaluating the quantity of classes in an ontology that do not possess definitions beyond its position in the hierarchy, it is possible to observe the level of incompleteness of each class and arrive to a conclusion about the completeness criteria.

The second criteria of the qualitative evaluation, *completeness*, was based on total of primitive classes, considered of simple descriptions, without detailed axioms. From 194 classes nominated, 166 are primitives. This corresponds to 85% of the classes in the ontology. Those 194 classes, only 57 classes are really vague in its description, representing 30% of the total of classes of the SwTO^I. The Table 3 presents this analysis. All of SwTO^I, 30% of the classes can be better

Classes Characteristics	Total	
Nominated Classes	194	
Primitive Classes	166	85%
Primitive Classes with vagueness description	57	**30%**

Table 3. Classes with vagueness description

represented. They are classes like *TestPattern*, *OSOnto:UserDocumentation* and *OSOnto:TechnicalDocumentation*, which were not detailed due to the initial work delimitation. However, those classes are subject to detailment.

Conciseness Criteria - The conciseness criteria of the ontology was analyzed of subjective form, considering the revisions, observations detached by the specialists of the research group and the reasoner conclusion concerning in redundancy verification between definitions terms and redundancies that can be inferred of others axioms.

To verify if the ontology had unnecessary definitions, the specialists of the research group carried out several revisions and detected some definitions as class *Author*, that represented the creators of a *software*, *hardware* or *virtual community*. Second the evaluation of the specialists, the creators of a *hardware* or *virtual community* are outside of the ontology purpose. During the development of the SwTO^I, SwTO and OSOnto the unnecessary definitions aimed by the specialists represented a total of 33% regarding all the definitions of the ontologies that properly were removed and in the final version, these unnecessary definitions are not present any more.

6 Conclusions

A formal representation of the knowledge supported by an ontology can aggregate many benefits for the Linux development process that is found in continuous evolution. Three important benefits can be noticeable. The first one is going to supply a formal domain vocabulary of the Linux test that represents the group consensus. The second one is the semantic record of the systematic tests elaboration criteria. The third one is the semantic record of the test designers knowledge. The benefits cited are aligned with the present needs of the Linux test community, which waits for contributions to optimize this process.

References

1. Costa, A.: Software Test Sequence Generation with the aid of Ontology, Master's Thesis, (2007), Federal University of Amazonas - UFAM,
2. Bezerra, D.: SwTO^I (Software Test Ontology Integrated): an Ontology with aplication in Linux Test, Master's Thesis, (2008), Federal University of Amazonas - UFAM,
3. Denaux, R.: Ontology-based Interactive User Modeling for Adaptative Web Information Systems Master's Thesis, (2005), Technische Universiteit Eindhoven
4. Horridge, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C.: A Practical Guide to Building OWL Ontologies using the Protégé-OWL Plugin and CO-0DE Tools Edition 1.0, (2004), University of Manchester.
5. Huang, N., Diao, S.: Structure-Based Ontology Evaluation. IAT '06: Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology (2006) 211-217 IEEE Computer Society Washington, DC, USA
6. Staab, S., Studer, R.: Handbook on Ontologies. Berlin: Springer-Verlag (2004)
7. Abran, A., Moore, J., Bourque, P., Dupuis, R., Tripp, L.: Guide to the Software Engineering Body of Knowledge - SWEBOK (2004) 1-202 IEEE Press Piscataway, NJ, USA
8. Uschold, M., Grüninger, M.: Ontologies: principles, methods, and applications (1996) 93-155 Jornal of Knowledge Engineering Review v.11, number 2.