Validating Ontologies with OOPS!

María Poveda-Villalón, Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez

Ontology Engineering Group. Departamento de Inteligencia Artificial.
Facultad de Informática, Universidad Politécnica de Madrid. Spain

{mpoveda, mcsuarez, asun}@fi.upm.es

Abstract. Ontology quality can be affected by the difficulties involved in ontology modelling which may imply the appearance of anomalies in ontologies. This situation leads to the need of validating ontologies, that is, assessing their quality and correctness. Ontology validation is a key activity in different ontology engineering scenarios such as development and selection. This paper contributes to the ontology validation activity by proposing a web-based tool, called OOPS!, independent of any ontology development environment, for detecting anomalies in ontologies. This tool will help developers to improve ontology quality by automatically detecting potential errors.

Keywords: ontology, pitfalls, ontology evaluation, ontology validation

1 Introduction

The emergence of ontology development methodologies during the last decades has facilitated major progress, transforming the art of building ontologies into an engineering activity. The correct application of such methodologies benefits the ontology quality. However, such quality is not always guaranteed because developers must tackle a wide range of difficulties and handicaps when modelling ontologies [1, 2, 11, 15]. These difficulties can imply the appearance of anomalies in ontologies. Therefore, ontology evaluation, which checks the technical quality of an ontology against a frame of reference [18], plays a key role in ontology engineering projects.

Ontology evaluation, which can be divided into validation and verification [18], is a complex ontology engineering process mainly due to two reasons. The first one is its applicability in different ontology engineering scenarios, such as development and reuse, and the second one is the abundant number of approaches and metrics [16].

One approach for validating ontologies is to analyze whether the ontology is conform to ontology modelling best practices; in other words, to check whether the ontologies contain anomalies or pitfalls. In this regard, a set of common errors made by developers during the ontology modelling is described in [15]. Moreover, in [10] a classification of errors identified during the evaluation of different features such as consistency, completeness, and conciseness in ontology taxonomies is provided. Finally, in [13] authors identify an initial catalogue of common pitfalls.

In addition, several tools have been developed to alleviate the dull task of evaluating ontologies. These tools support different approaches like (a) to check the consistency of the ontology, (b) to check the compliance with the ontology language used to

build the ontology or (c) to check modelling mistakes. In this context, our goal within this paper is to present an on-line tool that supports the automatic detection of pitfalls in ontologies. This tool is called OOPS! (OntOlogy Pitfall Scanner!) and represents a new option for ontology developers within the great range of ontology evaluation tools as it enlarges the list of errors detected by most recent and available works (like MoKi[1] [12] and XD Analyzer[2]). In addition, OOPS! is executed independently of the ontology development platform without configuration or installation and it also works with main web browsers (Firefox, Chrome, Safari and Internet Explorer[3]).

The remainder of this paper is structured as follows: Section 2 presents related work in ontology evaluation techniques and tools while Section 3 describes the pitfall catalogue taken as starting point in our evaluation approach. Section 4 shows OOPS! architecture and an illustrative use case. In Section 5 the user-based evaluation carried out over OOPS! is detailed. Finally, Section 6 outlines some conclusions and future steps to improve OOPS!.

2 State of the Art

In the last decades a huge amount of research on ontology evaluation has been performed. Some of these attempts have defined a generic quality evaluation framework [6, 9, 10, 17], other authors proposed to evaluate ontologies depending on the final (re)use of them [18], others have proposed quality models based on features, criteria and metrics [8, 3], and in recent times methods for pattern-based evaluation have also emerged [5, 14,]. A summary of generic guidelines and specific techniques for ontology evaluation can be found on [16].

Despite vast amounts of frameworks, criteria, and methods, ontology evaluation is still largely neglected by developers and practitioners. The result is many applications using ontologies following only minimal evaluation with an ontology editor, involving, at most, a syntax checking or reasoning test. Also, ontology practitioners could feel overwhelmed looking for the information required by ontology evaluation methods, and then, to give up the activity. That problem could stem from the time-consuming and tedious nature of evaluating the quality of an ontology.

To alleviate such a dull task technological support that automate as many steps involved in ontology evaluation as possible have emerged. In 2002 Fernández-López and Gómez-Pérez [7] developed ODEClean providing technological support to OntoClean Method [19] in the form of a plug-in for the WebODE ontology development environment. Few years later, ODEval[4] [4] was developed to help users evaluating RDF(S) and DAML+OIL concept taxonomies. Within those tools that support OWL ontologies we can find some developed as plug-ins for desktop applications as XDTools plug-in for NeOn Toolkit and OntoCheck plug-in for Protégé. This kind of

[1] https://moki.fbk.eu/website/index.php (Last visit on 14-04-2012)

[2] http://neon-toolkit.org/wiki/XDTools (Last visit on 14-04-2012)

[3] You may experience some layout strange behaviours with Internet Explorer.

[4] Even thought the approach described in the bibliographic documentation addresses OWL ontologies the on-line application available at http://oeg1.dia.fi.upm.es/odeval/ODEval.html only works with RDF(S) and DAML+OIL ontologies.

tools have two main disadvantages: (a) to force the user to install the ontology editor in which they are included as a plug-in and (b) to tend to be outdated, and sometimes incompatible, as long the core desktop applications evolve to new versions. Other tools rely on web based technologies as MoKi [12] that consists on a wiki-based ontology editor that incorporates ontology evaluation functionalities. In this case, although a testing user account is provided to try out the tool, an installation process is also required to set up the wiki system. Finally, command line tools like Eyeball[5] have been proposed. Eyeball is also available as Java API, what makes its use more suitable for users with technological background. In order to provided a more user-friendly version of Eyeball a graphical user interface is also provided, however it is still in an experimental phase.

As already mentioned in Section 1, different ontology evaluation tools can follow different approaches, and therefore check the ontology quality against different kind of issues. After performing an analysis of available tools we have realized that the following six dimensions[6] (Fig. 1) can be identified with respect to ontology quality:

- **Human understanding** dimension refers to whether the ontology provides enough information so that it can be understood from a human point of view. This aspect is highly related to the ontology documentation and clarity of the code.
- **Logical consistency** dimension refers to whether (a) there are logical inconsistencies or (b) there are parts of the ontology that could potentially lead to an inconsistency but they cannot be detected by a reasoner unless the ontology is populated.
- **Modelling issues** dimension refers to whether the ontology is defined using the primitives provided by ontology implementation languages in a correct way, or whether there are modelling decision that could be improved.

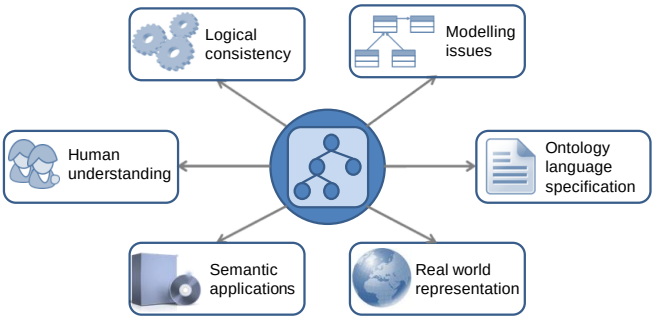

Fig. 1. Ontology Evaluation Dimensions

- **Ontology language specification** dimension refers to whether the ontology is compliant (e.g., syntax correctness) with the specifications of the ontology language used to implement the ontology.

[5] http://jena.sourceforge.net/Eyeball/

[6] It should be noted that this enumeration is not intended to be exhaustive and there could be more aspects to check an ontology against.

- **Real world representation** dimension refers to how accurately the ontology represents the domain intended for modelling. This dimension should be checked by humans (e.g., ontology engineers and domain experts).
- **Semantic applications** dimension refers to whether the ontology is fit for the software that uses it, for example checking availability, format compatibility, etc.

The results of the comparative analysis performed over available tools that support ontologies written in OWL (including OOPS!) with respect to the six aforementioned dimensions is shown in Table 1. This table presents the comparison according to (a) three general characteristics (whether they are IDE[7] independent, if a GUI[8] is provided or whether an installation process is needed) and (b) the ontology quality dimensions they consider, to some extent, into their algorithm. Ticks (✓) appearing in Table 1 mean that the given tool fulfils a general characteristic or it addresses a dimension; while crosses (×) mean that the tool does not fulfil the characteristic or does not provide any support for the given dimension. In this sense, we say that an ontology evaluation tool addresses a given dimension if it checks the ontology quality against at least one issue related to that dimension. For example, when a given tool checks whether the ontology contains individuals belonging to two disjoint classes, we can argue that this tool addresses the logical consistency dimension.

Table 1. Ontology evaluation tools comparison.

	XD-Tools	OntoCheck	EyeBall	Moki	**OOPS!**
General Characteristics					
IDE development independent	×	×	✓	×	✓
GUI provided	✓	✓	× (experimental)	✓	✓
No installing process required	×	×	×	×	✓
Ontology Evaluation Dimensions					
Human understanding	✓	✓	×	✓	✓
Logical consistency	✓	×	✓	×	✓
Modelling issues	✓	×	×	✓	✓
Ontology language specification	✓	×	✓	×	✓
Real world representation	×	×	×	×	✓
Semantic applications	×	×	×	×	×

3 Pitfall Catalogue so far

One of the crucial issues in ontology evaluation is the identification of anomalies or bad practices in the ontologies. As already mentioned in Section 1, different research works have been focused on establishing sets of common errors [15, 10, 11, 13].

Having the pitfall catalogue presented in [13] as starting point, we are performing a continuous process of maintenance, revision, and enlargement of such a catalogue as long as we discover new pitfalls during our research. Up to the moment of writing this paper, 5 new pitfalls have been included in the catalogue (P25-P29). Thus, the current

[7] Integrated Development Environment
[8] Graphical User Interface

version of the catalogue[9] consists on the 29 pitfalls shown in Table 2. Pitfalls in such a table are grouped by the quality dimensions presented in Section 2.

Table 2. Catalogue of pitfalls grouped by ontology quality dimension.

Human understanding	Modelling issues
• P1. Creating polysemous elements • P2. Creating synonyms as classes • P7. Merging different concepts in the same class • P8. Missing annotations • P11. Missing domain or range in properties • P12. Missing equivalent properties • P13. Missing inverse relationships • P19. Swapping intersection and union • P20. Misusing ontology annotations • P22. Using different naming criteria in the ontology	• P2. Creating synonyms as classes • P3. Creating the relationship "is" instead of using "rdfs:subClassOf", "rdf:type" or "owl:sameAs" • P4. Creating unconnected ontology elements • P5. Defining wrong inverse relationships • P6. Including cycles in the hierarchy • P7. Merging different concepts in the same class • P10. Missing disjointness • P17. Specializing too much a hierarchy • P11. Missing domain or range in properties • P12. Missing equivalent properties • P13. Missing inverse relationships • P14. Misusing "owl:allValuesFrom" • P15. Misusing "not some" and "some not" • P18. Specifying too much the domain or the range • P19. Swapping intersection and union • P21. Using a miscellaneous class • P23. Using incorrectly ontology elements • P24. Using recursive definition • P25. Defining a relationship inverse to itself • P26. Defining inverse relationships for a symmetric one • P27. Defining wrong equivalent relationships • P28. Defining wrong symmetric relationships • P29. Defining wrong transitive relationships
Logical consistency	
• P5. Defining wrong inverse relationships • P6. Including cycles in the hierarchy • P14. Misusing "owl:allValuesFrom" • P15. Misusing "not some" and "some not" • P18. Specifying too much the domain or the range • P19. Swapping intersection and union • P27. Defining wrong equivalent relationships • P28. Defining wrong symmetric relationships • P29. Defining wrong transitive relationships	
Real world representation	
• P9. Missing basic information • P10. Missing disjointness	

As part of the maintaining process of the pitfall catalogue, our intention is to extend it also with pitfalls proposed by users. Up to now, the suggestions from users are gathered using a form where they can describe what they consider to be a pitfall. In such a form, users can also add information about (a) how this suggested pitfall could be solved, (b) how important it could be to solve the pitfall when it appears, and (c) how it could be automatically detected. After a revision process the suggested pitfalls could be included in the catalogue.

4 OOPS!

OOPS! is a web-based tool, independent of any ontology development environment, for detecting potential pitfalls that could lead to modelling errors. This tool is intended to help ontology developers during the ontology validation activity [18], which can be divided into diagnosis and repair. Currently, OOPS! provides mechanisms to

[9] The official catalogue consists on the list of the pitfalls as well as their descriptions and it can be found at http://www.oeg-upm.net/oops/catalogue.jsp.

automatically detect as many pitfalls as possible, thus helps developers in the diagnosis activity. In the near future OOPS! will include also prescriptive methodological guidelines for repairing the detected pitfalls.

In this section we first explain the internal architecture of OOPS! and its main components (Section 4.1), followed by an exemplary use case showing how OOPS! helps ontology developers during the validation activity (Section 4.2).

4.1 How OOPS! is internally organized

Fig. 2 presents OOPS! underlying architecture in which it can be seen that OOPS! takes as input an ontology and the pitfall catalogue in order to produce a list of evaluation results. OOPS! is a web application based on Java EE[10], HTML[11], jQuery[12], JSP[13] and CSS[14] technologies. The web user interface consists on a single view where the user enters the URI pointing to or the RDF document describing the ontology to be analyzed. Once the ontology is parsed using the Jena API[15], it is scanned looking for pitfalls from those available in the pitfall catalogue (Section 3). During this scanning phase, the ontology elements involved in potential errors are detected; in addition, warnings regarding RDF syntax and some modelling suggestions are generated. Finally, the evaluation results are displayed by means of the web user interface showing the list of appearing pitfalls, if any, and the ontology elements affected as well as explanations describing the pitfalls.

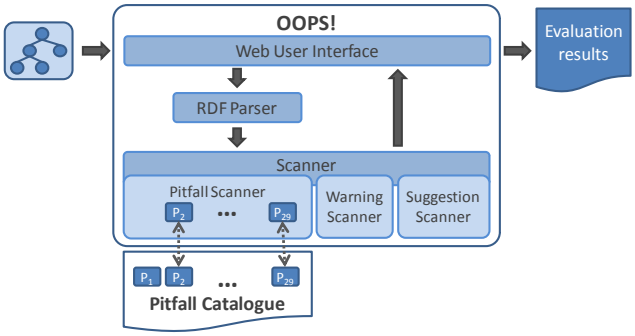

Fig. 2 OOPS! architecture

The "Pitfall Scanner" module, shown in Fig. 2, implements the automatic detection of a subset of 21 pitfalls of those included in the catalogue[16]. This subset includes pitfalls related to the following dimensions: (a) human understanding (P2, P7, P8, P11, P12, P13, P19, P20, and P22); (b) logical consistency (P5, P6, P19, P27, P28,

[10] http://www.oracle.com/technetwork/java/javaee/overview/index.html
[11] http://www.w3.org/html/wg/
[12] http://jquery.com/
[13] http://www.oracle.com/technetwork/java/javaee/jsp/index.html
[14] http://www.w3.org/Style/CSS/
[15] http://jena.sourceforge.net/
[16] http://www.oeg-upm.net/oops/catalogue.jsp

and P29); (c) real world representation (P10); and (d) modelling issues (P2, P3, P4, P5, P6, P7, P10, P11, P12, P13, P19, P21, P24, P25, P26, P27, P28, and P29). It is worth mentioning that since the catalogue consists on a list of pitfalls defined in natural language, they have to be transformed into a formal or procedural language in order to detect them automatically. Currently, this transformation is implemented in OOPS! as a Java class for each of the 21 pitfalls. In order to detect a greater range of pitfalls, developers should implement the appropriate Java class and plug it into the Pitfall Scanner" module. Up to now, OOPS! provides an on-line form[17] where users can suggest new pitfalls by describing them in natural language and attaching diagrams if needed. Once a pitfall suggestion is reviewed and accepted, it can be included in OOPS! by implementing the corresponding Java class as already mentioned.

The automatic detection of pitfalls has been approached in two different ways. On the one hand, some pitfalls (namely P3, P7, P12, P20, P21, and P22) have been automated by checking general characteristics of the ontology, for example the use of more than one naming convention to detect P22 (Using different naming criteria in the ontology). On the other hand, other pitfalls (namely P2, P4, P5, P6, P8, P10, P11, P13, P19, P24, P25, P26, P27, P28, and P29), related to the internal structure of the ontology, have been translated into patterns that indicate that a pitfall appears when a pattern is spotted. Fig. 3 shows some of the patterns used to detect pitfalls within OOPS!; for example, the pattern used to detect P5 (Defining wrong inverse relationships) consist on pairs of relationships defined as inverse but where the domain of one of them is not equal to the range of the other. Up to now, these patterns are spotted programmatically by means of a Java class; however, our aim is to transform into SPARQL[18] queries as many patterns as possible in future releases of OOPS!.

The module "Warning Scanner" identifies cases where a class or property is not defined as such by means of the corresponding OWL primitive, that is, related to the "ontology language specification" dimension presented in Section 2. It is worth mentioning that currently there is not a Java class looking for all the cases within the ontology. Instead, these warnings are spotted on running time during the execution of the "Pitfall Scanner" module so that only the classes and relationships related to the other pitfalls detection are flag up.

Finally, the module "Suggestion Scanner" looks for properties with equal domain and range axioms and proposes them as potential symmetric or transitive properties.

4.2 How OOPS! works

OOPS! main functionality is to analyze ontologies[19] (a) via the URI in which an ontology is located or (b) via text input containing the source code of the ontology. As a result of the analysis, OOPS! informs developers about which elements of the ontology are possibly affected by pitfalls.

[17] http://www.oeg-upm.net/oops/submissions.jsp

[18] http://www.w3.org/TR/rdf-sparql-query/

[19] The input ontology must be implemented in OWL (http://www.w3.org/TR/2009/REC-owl2-primer-20091027/) or RDF (http://www.w3.org/TR/2004/REC-rdf-primer-20040210/).

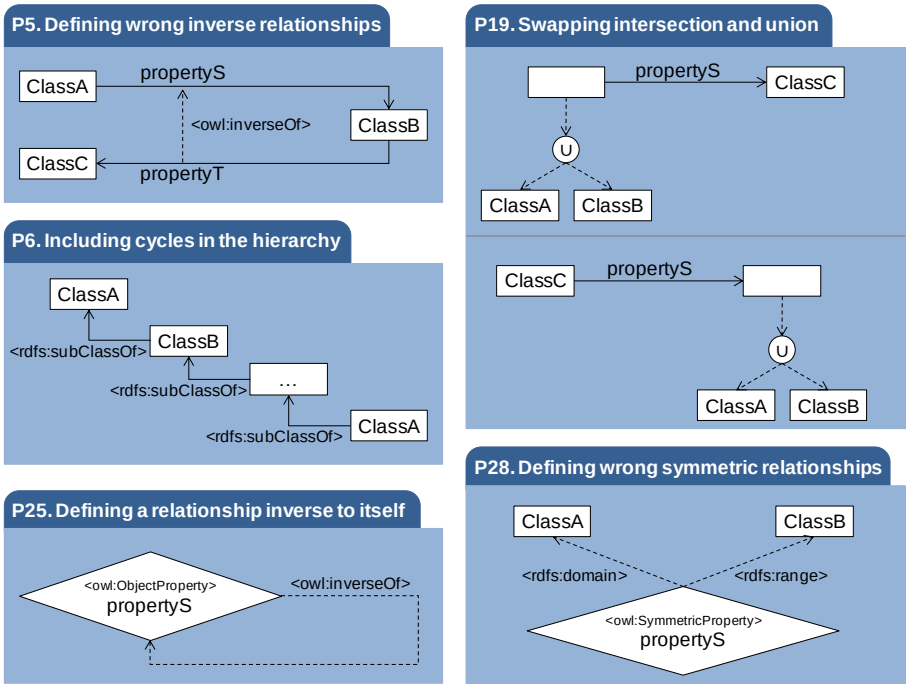

Fig. 3. Example of patterns defined to detect pitfalls

Fig. 4 shows OOPS! home page[20] where a user can enter an ontology to be analyzed via URI or RDF coding. This page also presents a brief description of OOPS!. In addition, the menu in the right side of the web site contains links to (a) documentation related to OOPS! (the pitfall catalogue, a user guide, and a technical report) and (b) papers related to OOPS! and the research behind it. In addition, two different ways in which users can send their suggestion are also provided: (1) a questionnaire to send feedback after using the tool and (2) a form to suggest new pitfalls.

As result of analyzing the ontology provided by the user, OOPS! generates, as it is shown in Fig. 5[21], a new web page listing the appearing pitfalls in the ontology. This list provides information about (a) how many times a particular pitfall appears, (b) which specific ontology elements are affected by such a pitfall, and (c) a brief description about what the pitfall consist on.

It is worth mentioning that OOPS! output points to ontology elements identified as potential errors but they are not always factual errors as sometimes something can be considered a factual errors depending on different perspectives (such as the particular ontology being analyzed, its particular requirements, the characteristics of the domain intended for modelling, etc.). In this sense, there are seven pitfalls (P3, P8, P22, P25, P26, P28, and P29) that should be repaired when detected by OOPS! as they certainly

[20] http://www.oeg-upm.net/oops

[21] For the sake of clarity some screenshots have been reduced keeping the interesting information for each example.

point to modelling problems within the ontology; while the rest of pitfalls appearing in OOPS! output must be manually checked in order to discern whether they point to factual problems in the ontology being analyzed.

Fig. 4 OOPS! home page

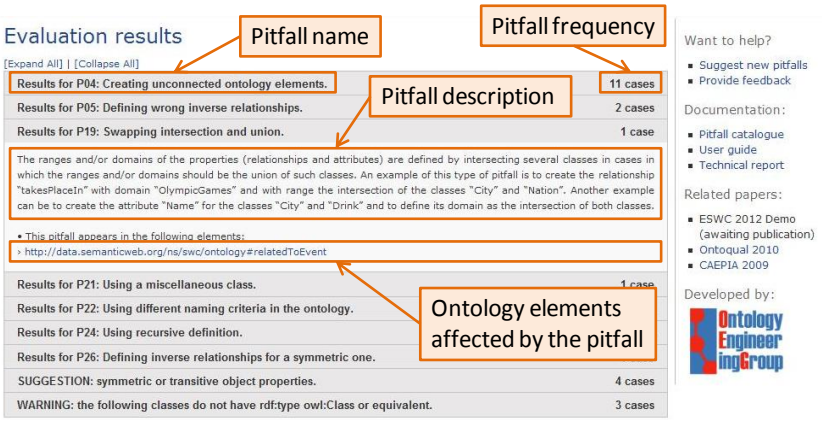

Fig. 5 Example of evaluation results web page generated by OOPS!

OOPS! shows the result for each pitfall in three different ways depending on the kind of pitfall. There are pitfalls that affect individual elements in the ontology, others affect more than one element, and there are also pitfalls that do not affect particular ontology elements but the whole ontology. In the following, an example of OOPS!

execution is shown in order to clarify the different types of results a user can get from OOPS! and how to interpret them. For illustrating the example we are using the Semantic Web Conference Ontology[22] (SWC).

After executing OOPS! with the SWC ontology, we obtain a summary of the pitfalls encountered as presented in Fig. 6 . Such a figure shows that 11 pitfalls have been detected as well as 1 suggestion and 1 warning. For the sake of simplicity, not all results will be detailed but only those contributing to an explanation of different kind of outputs or interpretations.

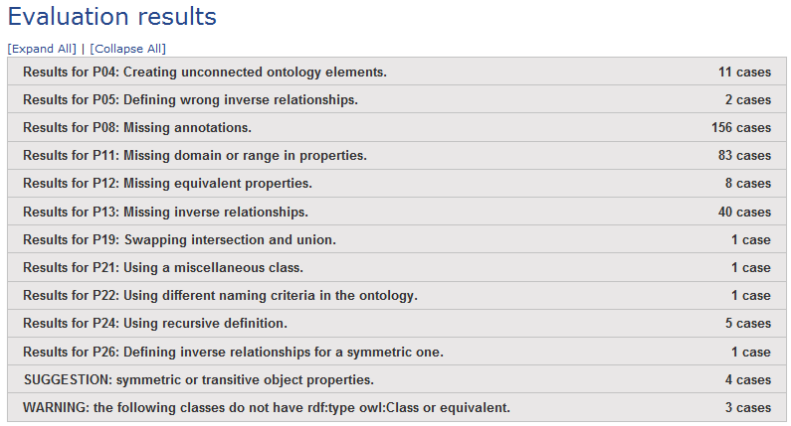

Fig. 6 Evaluation results for SWC ontology

As already mentioned, some pitfalls can affect individual ontology elements, other pitfalls can affect more than one element in the ontology, and others can affect the whole ontology. Fig. 7 shows an example[23] of a pitfall (P08. Missing annotations) that affects individual ontology elements. In this case, the output is grouped by (a) elements that have neither `rdfs:label` or `rdfs:comment` defined and (b) elements that have no `rdfs:comment`.

Fig. 8 shows an example of a pitfall (P05. Defining wrong inverse relationships) that affects more than one ontology element. In this case, when OOPS! detects a potential mistake while defining inverse relationships it provides the pair of relationships involved in such pitfall.

Fig. 9 shows a particular example of pitfall (P22. Using different naming criteria in the ontology), which affects the whole ontology. It is worth mentioning that the ontology elements shown in Fig. 9 represent just arbitrary example as P22 points to the

[22] Official URI is http://data.semanticweb.org/ns/swc/swc_2009-05-09.rdf. As the results shown in this paper may be affected by possible updates on the original ontology there is a copy of the ontology code used in this example on 18-04-2012 that can be found at http://www.oeg-upm.net/files/mpoveda/EKAW2012/swc_2009-05-09.rdf.

[23] As it is shown in the top part of the example there have been found 156 cases of this pitfall, however just an excerpt of the results is shown due to space constraints. This case may also apply to further examples.

fact of having different naming criteria along the ontology instead of between particular elements.

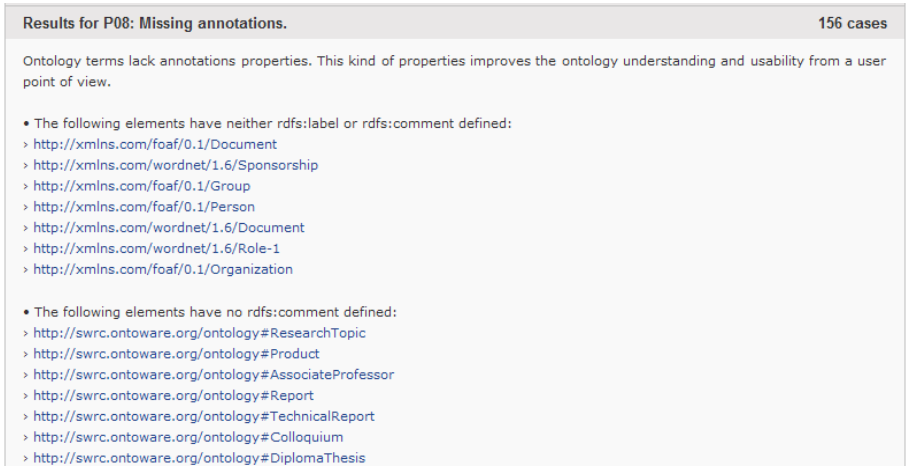

Fig. 7 Excerpt of an example of evaluation results for P08: Missing annotations

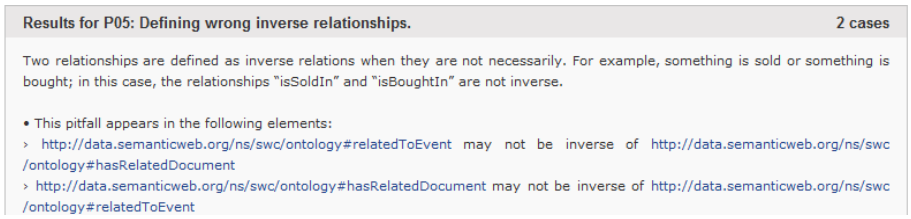

Fig. 8 Example of evaluation results for P05: Defining wrong inverse relationships

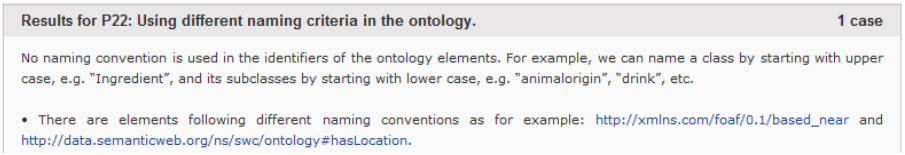

Fig. 9 Example of evaluation results for P22: Using different naming criteria in the ontology

5 User-based Evaluation of OOPS!

OOPS! has been used in different research and educational projects with positive feedback and interesting comments from the developers involved in each case. In this section we briefly summarize a set of such cases, presenting qualitative results[24].

[24] Quantitative results are not provided because to test the same real case using the proposed tool and without the tool was not feasible due to the effort needed.

The first case we can mention is the use of OOPS! in the context of two Spanish research projects called *mIO!*[25] and *Buscamedia*[26]. Both projects involved the development of two ontologies about user context in mobile environments and multimedia information objects respectively. In both projects the validation activity was carried out using OOPS!. After the diagnosis phase, the ontologies were repaired accordingly to OOPS! output. It is worth mentioning that by the time when the validation activities were carried out (September 2011) OOPS! did not provide a graphical user interface, but it was provided to ontology developers involved in the projects as a .jar file and its output was given in a .txt file.

A total of seven ontology developers were involved in the ontology validation activities within mIO! and Buscamedia use cases. Such developers provided positive feedback about (a) OOPS! usefulness, (b) the advantages of being IDE independent and (c) coverage of pitfalls detected by OOPS! in comparison with other tools. They also provided very valuable feedback about aspects that could be improved in OOPS! as for example (a) providing a graphical user interface, (b) providing more information about what the pitfalls consist on, and (c) considering the imported ontologies during the analysis. All these comments were taking into account and implemented in subsequent releases of OOPS!. Other suggestions as (a) to allow the evaluation of subsets of pitfalls and (b) to provide some recommendations to repair the pitfalls found within the ontology are currently considered as future work to be included in next releases.

The second case refers to a controlled experiment to test the benefits of using OOPS! during the ontology validation activity that was carried out with master students. This experiment was performed during the ATHENS course that took place in November 2011 at *Universidad Politécnica de Madrid*. Twelve master students working in pairs executed the experiment. Before the experiment, students were provided with (a) some explanations about OOPS! and ontology evaluation concepts, (b) the detailed tasks to be performed during the experiment, and (c) two different ontologies to be evaluated. After the experiment, we gathered students' feedback using questionnaires. Most of the students considered that OOPS! was very useful to evaluate the ontologies at hand and that its output shows clearly what are the problems detected and in which elements. Again in this experiment, main proposed feature to be added in OOPS! was to include guidelines about how to solve pitfalls. In addition, some of the students commented that it could be useful (a) to associate colours to the output indicating how critical the pitfalls are, like error and warnings recognition in many software IDEs and (b) to provide a way to see the lines of the file that the error considered is originated from.

Finally, we announced the release of the tool thorough several mailing lists[27] related to the Semantic Web so that all the people interested in analyzing their ontologies can use OOPS! and send feedback after that. Up to now we have received four feedback responses from users not related to any project or any controlled experiment. This feedback shows that even though all of the users think that OOPS! is very useful, three of them will always use it within their ontology developments or recommend it

[25] http://www.cenitmio.es/

[26] http://www.cenitbuscamedia.es/

[27] For example http://lists.w3.org/Archives/Public/semantic-web/2012Mar/0064.html

to a colleague while one of them will do sometimes. Also some strengths of the tool were explicitly pointed out by users as[28]: "*easy to use*", "*no installation required*", "*quick results*" and "*good to detect low level problems in ontologies*". However, the richest side of this feedback is the set of proposals to improve the tool. The most important feedback in this regard refers to (a) show which pitfalls do not appear in the ontology, (b) include reasoning processes so that OOPS! would not complain when a property inherits domain/range from its superproperty, and (c) allow the evaluation of subsets of pitfalls.

Apart from the feedback received through the web questionnaire, we have also received comments and questions about OOPS! by email, what reveals users willingness to adopt this type of tools within their ontology developments. Within these feedback emails users also pointed out the need of developing systems like OOPS! in the context of ontological engineering. In addition, the following improvements for our tool were received: (a) to discard pitfalls involving terms properly marked as DEPRECATED following the OWL 2 deprecation pattern, (b) to take into account the different namespaces used within the ontology, (c) to look for URI misuse as using the same URI as two types of ontology elements, and (d) to look for non-standard characters in natural language annotations.

6 Conclusions and Future Work

Ontology evaluation is a key ontology engineering activity that can be performed following a variety of approaches and using different tools. In this paper we present (a) an evaluation approach based on the detection of anomalies in ontologies and (b) an on-line tool called OOPS! that automatically detects a subset of pitfalls from those gathered in a pitfall catalogue.

OOPS! represents a step forward within ontology evaluation tools as (a) it enlarges the list of errors detected by most recent and available works (e.g. MoKi [12] and XD Analyzer), (b) it is fully independent of any ontology development environment, and (c) it works with main web browsers (Firefox, Chrome, Safari, and Internet Explorer).

OOPS! has been tested in different settings (research and educational projects) with positive feedback from the ontology developers who evaluated their ontologies. Such ontology developers provided also interesting suggestions to improve the tool. Both feedback and suggestions have been provided via the feedback questionnaire available in OOPS! website. Apart from particular projects, OOPS! has been already used by other ontology developers who belong to different organizations (such as AtoS, Tecnalia, *Departament Arquitectura La Salle* at *Universitat Ramon Llull*, and Human Mobility and Technology Laboratory at CICtourGUNE). In fact, OOPS! is freely available to users on the Web. In addition, OOPS! is currently being tested by Ontology Engineering Group[29] members in order to debug it and extend its functionality.

As part of the continuous process of improving OOPS!, currently we are working in the improvement of the rules applied to automatically detect pitfalls to make them

[28] The following comments have been taken literally from the feedback questionnaires.

[29] http://www.oeg-upm.net/

more accurate. In addition, we are also working on the maintenance of the pitfall catalogue. For this purpose, OOPS! web site includes a form to suggest new pitfalls, what allows extending the catalogue in a collaborative way. In addition, as long as we discover new pitfalls during our research, they will be included in the current catalogue. New releases of OOPS! could include the detection of both new pitfalls proposed by users and new errors identified by us. In this regard, more ambitious plans are (a) the development of a formal language to define pitfalls so that a user can define in a formal way the pitfalls to be detected, and (b) the implementation of mechanisms so that OOPS! can automatically interpret and process the defined pitfalls without encoding them manually.

Based on the feedback we have already received, we have planned to improve OOPS by allowing the validation of groups of pitfalls the user is interested in. To do this, we are going to classify pitfalls in different categories according to the ontology quality criteria identified in [9] and [10]. This new feature will provide more flexibility to the ontology validation, since it will allow users to diagnose their ontologies just with respect to the dimensions or pitfalls they are interested in.

Regarding increasing OOPS! features to help the user in the activity of repairing the detected pitfalls, our plan is to provide more detailed descriptions about the pitfalls and some prescriptive methodological guidelines about how to solve them. Our plans also include associating priority levels to each pitfall according to the different types of consequences they can convey when appearing in an ontology. This feature will be useful to prioritize actions to be taken during the repairing task.

Finally, future plans also include making REST services available in order to allow other developments to use and integrate the pitfall scanner functionalities within their applications.

Acknowledgments. This work has been partially supported by the Spanish projects *BabelData* (TIN2010-17550) and *BuscaMedia* (CENIT 2009-1026).

References

1. Aguado de Cea, G., Gómez-Pérez, A., Montiel-Ponsoda, E., Suárez-Figueroa, M.C. *Natural language-based approach for helping in the reuse of ontology design patterns*. In Knowledge Engineering: Practice and Patterns, Proceedings of EKAW 2008, LNCS 5268, pp. 32–47, 2008.
2. Blomqvist, E., Gangemi, A., Presutti, V. *Experiments on Pattern-based Ontology Design*. In Proceedings of K-CAP 2009, pp. 41-48. 2009.
3. Burton-Jones, A., Storey, V.C., and Sugumaran, V., and Ahluwalia, P. *A Semiotic Metrics Suite for Assessing the Quality of Ontologies*. Data and Knowledge Engineering, (55:1) 2005, pp. 84-102.
4. Corcho, O., Gómez-Pérez, A., González-Cabero, R., Suárez-Figueroa, M.C. *ODEval: a Tool for Evaluating RDF(S), DAML+OIL, and OWL Concept Taxonomies*. In: 1st IFIP Conference on Artificial Intelligence Applications and Innovations (AIAI 2004), August 22-27, 2004, Toulouse, FRANCE
5. Djedidi, R., Aufaure, M.A. *Onto-Evoal an Ontology Evolution Approach Guided by Pattern Modelling and Quality Evaluation*, Proceedings of the Sixth International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2010), February 15-19 2010, Sofia, Bulgaria.

6. Duque-Ramos, A., López, U., Fernández-Breis, J. T., Stevens, R. *A SQUaRE-based Quality Evaluation Framework for Ontologies.* OntoQual 2010 - Workshop on Ontology Quality at the 17th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2010). ISBN: ISSN 1613-0073. CEUR Workshop Proceedings. Pages: 13-24. 15 October 2010. Lisbon, Portugal.
7. Fernández-López, M., Gómez-Pérez, A. *The integration of OntoClean in WebODE OntoClean method*. In Proceedings of the Evaluation of Ontology based Tools Workshop EON2002 at 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02). Spain 2002.
8. Flemming, A. *Assessing the quality of a Linked Data source*. Proposal. http://www2.informatik.hu-berlin.de/~flemming/Proposal.pdf
9. Gangemi, A., Catenacci, C., Ciaramita, M., Lehmann J. *Modelling Ontology Evaluation* and *Validation.* Proceedings of the 3rd European Semantic Web Conference (ESWC2006), number 4011 in LNCS, Budva. 2006
10. Gómez-Pérez, A. *Ontology Evaluation*. Handbook on Ontologies. S. Staab and R. Studer Editors. Springer. International Handbooks on Information Systems. Pp: 251-274. 2004.
11. Noy, N.F., McGuinness. D. L. *Ontology development 101: A guide to creating your first ontology.*Technical Report SMI-2001-0880, Standford Medical Informatics. 2001.
12. Pammer, V. *PhD Thesis: Automatic Support for Ontology Evaluation Review of Entailed Statements and Assertional Effects for OWL Ontologies*. Engineering Sciences. Graz University of Technology.
13. Poveda, M., Suárez-Figueroa, M.C., Gómez-Pérez, A. *A Double Classification of Common Pitfalls in Ontologies.* OntoQual 2010 - Workshop on Ontology Quality at the 17th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2010). ISBN: ISSN 1613-0073. CEUR Workshop Proceedings. Pages: 1-12. 15 October 2010. Lisbon, Portugal.
14. Presutti, V., Gangemi, A., David S., Aguado, G., Suárez-Figueroa, M.C., Montiel-Ponsoda, E., Poveda, M. *NeOn D2.5.1: A Library of Ontology Design Patterns: reusable solutions for collaborative design of networked ontologies*. NeOn project. (FP6-27595). 2008.
15. Rector, A., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C. *Owl pizzas: Practical experience of teaching owl-dl: Common errors and common patterns*. In Proc. of EKAW 2004, pp: 63–81. Springer. 2004.
16. Sabou, M., Fernandez, M. *Ontology (Network) Evaluation*. Ontology Engineering in a Networked World. Suárez-Figueroa, M.C., Gómez-Pérez, A., Motta, E., Gangemi, A. Editors. Pp. 193-212, Springer. 2012. ISBN 978-3-642-24793-4
17. Strasunskas, D., Tomassen, S.L. *The role of ontology in enhancing semantic searches: the EvOQS framework and its initial validation*. Int. J. Knowledge and Learning, Vol. 4, No. 4, pp. 398-414.
18. Suárez-Figueroa, M.C. *PhD Thesis: NeOn Methodology for Building Ontology Networks: Specification, Scheduling and Reuse.* Spain. Universidad Politécnica de Madrid. June 2010.
19. Welty, C., Guarino, N. *Supporting Ontological Analysis of Taxonomic Relationships*. Data and Knowledge Engineering. September 2001.