OOPS! – OntOlogy Pitfalls Scanner!

María Poveda-Villalón and Mari Carmen Suárez-Figueroa
Ontology Engineering Group. Departamento de Inteligencia Artificial.
Facultad de Informática, Universidad Politécnica de Madrid.
Campus de Montegancedo s/n.
28660 Boadilla del Monte. Madrid. Spain
{mpoveda, mcsuarez}@fi.upm.es

Abstract. The application of methodologies for building ontologies has improved the ontology quality. However, such a quality is not totally guaranteed because of the difficulties involved in ontology modelling. These difficulties are related to the inclusion of anomalies or worst practices in the modelling. Several authors have provided lists of typical anomalies detected in ontologies during the last decade. In this context, our aim in this technical report is to describe OOPS! (OntOlogy Pitfalls Scanner!), a tool for pitfalls detection in ontology developments.

Keywords: pitfalls, worst practices, ontology evaluation, ontology engineering

1 Introduction

The 1990s and the first years of this new millennium have witnessed the growing interest of many practitioners in methodologies that support the creation of ontologies. All these approaches have supposed a step forward since they have transformed the art of building ontologies into an engineering activity. The correct application of such methodologies benefits the ontology quality. However, such a quality is not totally guaranteed because developers must tackle a wide range of difficulties and handicaps when modelling ontologies [1, 2, 4, 6]. These difficulties can imply the appearance of anomalies or worst practices in ontologies. Therefore, it is important to evaluate the ontologies before using or reusing them in other ontologies and/or semantic applications.

One of the crucial issues in ontology evaluation is the identification of anomalies or worst practices in the ontologies. In this regard, it is worth mentioning that in [6] the authors describe a set of common errors made by developers during the ontology modelling. Moreover, in [3] a classification of errors identified during the evaluation of consistency, completeness, and conciseness of ontology taxonomies is provided. Finally, in [5] authors identify an initial catalogue of common pitfalls[1].

In this context, our main goal is to provide an automated tool to help the ontology practitioners detecting common pitfalls during the ontology development. It is worth to note that OOPS![2] can detect some of the pitfalls in a automated way and others must be checked manually as they are detected in a semi-automated fashion. .

[1] http://www.oeg-upm.net/oops/catalogue.jsp
[2] http://www.oeg-upm.net/oops/

The remainder of this report is structured as follows: Section 2 presents the catalogue of pitfalls which detection is (semi)-automated in OOPS! while Section 3 describes main OOPS! implemented features up to the moment of writing this document. Finally, Section 4 describes OOPS! architecture.

2 Catalogue of Common Pitfalls

In this section the catalogue of pitfalls that OOPS! proposed in [5] as well as some extensions to such catalogue are presented. This catalogue consists of 29 pitfalls and includes the pitfall definitions, that is, what is the anomaly, as well as some related examples when possible.

P1. **Creating polysemous elements:** an ontology element whose name has different meanings is included in the ontology to represent more than one conceptual idea. For example, the class "Theatre" is used to represent both the artistic discipline and the place in which a play is performed.

P2. **Creating synonyms as classes:** several classes whose identifiers are synonyms are created and defined as equivalent. As an example we could define "Car", "Motorcar" and "Automobile" as equivalent classes. Another example is to define the classes "Waterfall" and "Cascade" as equivalents. This pitfall is related to the guidelines presented in [4] which explain that synonyms for the same concept do not represent different classes.

P3. **Creating the relationship "is" instead of using "subclassOf", "instanceOf" or "sameIndividual":** the "is" relationship is created in the ontology instead of using OWL primitives for representing the subclass relationship ("subclassOf"), the membership to a class ("instanceOf"), or the equality between instances ("sameAs"). An example of this type of pitfall is to define the class "Actor" in the following way 'Actor ≡ Person ⊓ ∃interprets.Actuation ⊓ ∃is.Man'. This pitfall is related to the guidelines for understanding the "is-a" relation provided in [4].

P4. **Creating unconnected ontology elements:** ontology elements (classes, relationships or attributes) are created with no relation to the rest of the ontology. An example of this type of pitfall is to create the relationship "memberOfTeam" and to miss the class representing teams; thus, the relationship created is isolated in the ontology.

P5. **Defining wrong inverse relationships:** two relationships are defined as inverse relations when actually they are not. For example, something is sold or something is bought; in this case, the relationships "isSoldIn" and "isBoughtIn" are not inverse.

P6. **Including cycles in the hierarchy** [3, 4]**:** a cycle between two classes in the hierarchy is included in the ontology, although it is not intended to have such classes as equivalent. That is, some class A has a subclass B and at the same time B is a superclass of A. An example of this type of pitfall is represented by the class "Professor" as subclass of "Person", and the class "Person" as subclass of "Professor".

P7. **Merging different concepts in the same class:** a class is created whose identifier is referring to two or more different concepts. An example of this type of pitfall is to create the class "StyleAndPeriod", or "ProductOrService".

P8. **Missing annotations:** ontology terms lack annotations properties. This kind of properties improves the ontology understanding and usability from a user point of view.

P9. **Missing basic information:** needed information is not included in the ontology. Sometimes this pitfall is related with the requirements in the ORSD [7, 8] that are not covered by the ontology. Other times it is related with knowledge that could be added to the ontology in order to make it more complete. An example of this type of pitfall is to create the relationship "startsIn" to represent that the routes have a starting point in a particular location; and to miss the relationship "endsIn" to show that a route has an end point. Another example is to create the relationship "follows" when modelling order relations; and do not create its inverse relationship "precedes".

P10. **Missing disjointness** [3, 4, 6]**:** the ontology lacks disjoint axioms between classes or between properties that should be defined as disjoint. For example, we can create the classes "Odd" and "Even" (or the classes "Prime" and "Composite") without being disjoint; such representation is not correct based on the definition of these types of numbers.

P11. **Missing domain or range in properties:** relationships and/or attributes without domain or range (or none of them) are included in the ontology. There are situations in which the relation is very general and the range should be the most general concept "Thing". However, in other cases, the relations are more specific and it could be a good practice to specify its domain and/or range. An example of this type of pitfall is to create the relationship "hasWritten" in an ontology about art in which the relationship domain should be "Writer" and the relationship range should be "LiteraryWork". This pitfall is related to the common error when defining ranges and domains described in [6].

P12. **Missing equivalent properties:** when an ontology is imported into another, classes that are duplicated in both ontologies are normally defined as equivalent classes. However, the ontology developer misses the definition of equivalent properties in those cases of duplicated relationships and attributes. For example, the classes "CITY" and "City" in two different ontologies are defined as equivalent classes; however, relationships "hasMember" and "has-Member" in two different ontologies are not defined as equivalent relations.

P13. **Missing inverse relationships:** there are two relationships in the ontology that should be defined as inverse relations. For example, the case in which the ontology developer omits the inverse definition between the relations "hasLanguageCode" and "isCodeOf", or between "hasReferee" and "isRefereeOf".

P14. **Misusing "allValuesFrom"** [6]**:** this pitfall can appear in two different ways. In the first, the anomaly is to use the universal restriction ("allValuesFrom") as the default qualifier instead of using the existential restriction ("someValuesFrom"). This means that the developer thinks that "allValuesFrom" implies "someValuesFrom". In the second, the mistake is to include "allValuesFrom" to close off the possibility of further additions for a given property. An example of this type of pitfall is to define the class "Book" in the following way 'Book ≡ ∃producedBy.Writer ⊓ ∀uses.Paper' and closing the possibility of adding "Ink" as an element used in the writing.

P15. **Misusing "not some" and "some not"** [6]**:** to mistake the representation of "some not" for "not some", or the other way round. An example of this type of pitfall is to define a vegetarian pizza as any pizza which both has *some* topping which is *not* meat and also has *some* topping which is *not* fish. This example is explained in more detail in [6].

P16. **Misusing primitive and defined classes** [6]**:** to fail to make the definition 'complete' rather than 'partial' (or 'necessary and sufficient' rather than just 'necessary). It is critical to understand that, in general, nothing will be inferred to be subsumed under a primitive class by the classifier. This pitfall implies that the developer does not understand the open world assumption. A more detailed explanation and examples can be found in [6].

P17. **Specializing too much a hierarchy:** the hierarchy in the ontology is specialized in such a way that the final leaves cannot have instances, because they are actually instances and should have been created in this way instead of being created as classes. Authors in [4] provide guidelines for distinguishing between a class and an instance when modelling hierarchies. An example of this type of pitfall is to create the class "RatingOfRestaurants" and the classes "1fork", "2forks", and so on, as subclasses instead of as instances. Another example is to create the classes "Madrid", "Barcelona", "Sevilla", and so on as subclasses of "Place". This pitfall could be also named "Individuals are not Classes".

P18. **Specifying too much the domain or the range** [4, 6]**:** not to find a domain or a range that is general enough. An example of this type of pitfall is to restrict the domain of the relationship "isOfficialLanguage" to the class "City", instead of allowing also the class "Country" to have official language or a more general concept such as "GeopoliticalObject".

P19. **Swapping intersection and union:** the ranges and/or domains of the properties (relationships and attributes) are defined by intersecting several classes in cases in which the ranges and/or domains should be the union of such classes. An example of this type of pitfall is to create the relationship "takesPlaceIn" with domain "OlympicGames" and with range the intersection of the classes "City" and "Nation". Another example can be to create the attribute "Name" for the classes "City" and "Drink" and to define its domain as the intersection of both classes. This pitfall is related to the common error that appears when defining ranges and domains described in [6] and also related to the guidelines for defining these elements provided in [4].

P20. **Swapping Label and Comment:** the contents of the Label and Comment annotation properties are swapped. An example of this type of pitfall is to include in the Label annotation of the class "Crossroads" the following sentence 'the place of intersection of two or more roads'; and to include in the Comment annotation the word 'Crossroads'.

P21. **Using a miscellaneous class:** to create in a hierarchy a class that contains the instances that do not belong to the sibling classes instead of classifying such instances as instances of the class in the upper level of the hierarchy. This class is normally named "Other" or "Miscellaneous". An example of this type of pitfall is to create the class “HydrographicalResource”, and the subclasses “Stream”, “Waterfall”, etc., and also the subclass “OtherRiverElement”.

P22. **Using different naming criteria in the ontology:** no naming convention is used in the identifiers of the ontology elements. Some notions about naming conventions are provided in [4]. For example, we can name a class by starting with upper case, e.g. “Ingredient”, and its subclasses by starting with lower case, e.g. “animalorigin”, “drink”, etc.

P23. **Using incorrectly ontology elements:** an ontology element (class, relationship or attribute) is used to model a part of the ontology that should be modelled with a different element. A particular case of this pitfall regarding to the misuse of classes and property values is addressed in [4]. An example of this type of pitfall is to create the relationship “isEcological” between an instance of “Car” and the instance “Yes” or “No”, instead of creating the attribute “isEcological” whose range is Boolean.

P24. **Using recursive definition:** an ontology element is used in its own definition. For example, it is used to create the relationship “hasFork” and to establish as its range the following ’the set of restaurants that have at least one value for the relationship “hasFork”.

P25. **Defining a relationship inverse to itself:** an object property is defined as its own inverse object property. In this case, this property could have been defined as “owl:SymmetricProperty” instead. An example of this type of pitfall is to create the relationship “hasBorderWith” and to state that “hasBorderWith” is its inverse relationship.

P26. **Defining inverse relationships for a symmetric one:** an object property is defined as “owl:SymmetricProperty” and there is also an object property (it could be itself or another relationship) defined as its inverse. For example, the symmetric relationship “farFrom” has an inverse relationships defined, e.g. itself, “farFrom”.

P27. **Defining wrong equivalent relationships:** two relationships are defined as equivalent relations when they are not necessarily. For example, we can mix up common relationships that could hold between several types of entities, as "hasPart" defined between human body parts and the same relationship relating research plans as part of research projects.

P28. **Defining symmetric relationships that do not have same domain and range:** the domain defined for a symmetric relationship is different from its range. This could happen because the relationship might not be symmetric, for example defining the relation "pastProject" between the concepts "Agent" and "Project". This situation can also appear due to the domain and range are too specified, for example, if we define the symmetric relationship "hasSpouse" between the concepts "Man" and "Woman" instead of using the concept "Person" both as domain and range of such a relationship.

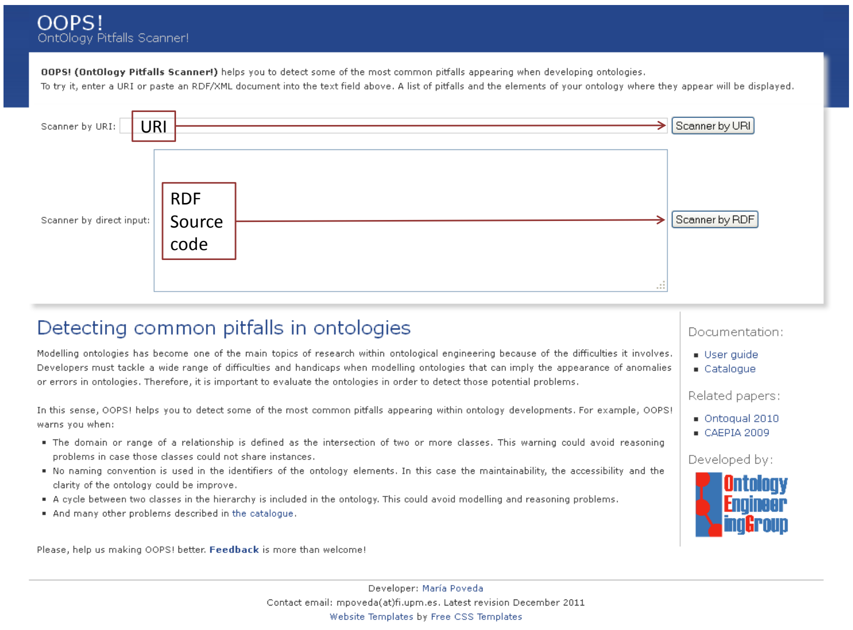
P29. **Defining transitive relationships that do not have same domain and range:** the domain defined for a transitive relationship is different from its range. An example of this type of error is to create the relationship "participatesIn", which domain is the union of the concepts "Team" and "Individual" and which range is the concept "Event", defining the relationship as transitive.

It is worth mentioning that we are working on the extension of the pitfall catalogue by means of including 1) new pitfalls, 2) methodological guidelines to avoid the pitfalls, that is, the solution to these ontology anomalies and 3) examples of the advantages of avoiding or correcting the pitfalls. In addition, the pitfalls have been grouped by two different evaluation approaches [5] and the ongoing work aims to classify them according to the different kind of consequences that could entail their appearance.

3 OOPS! features

OOPS! is a tool that scans ontologies looking for potential pitfalls that could led to modeling errors. OOPS! is very useful for ontology developers during the ontology validation activity, concretely during the diagnosis phase. Its main functionality is to analyze ontologies via URL or RDF coding and to inform developers about which elements of the ontology are possibly affected by pitfalls or syntax errors. It also provides modelling suggestions for some relationships. Up to the moment of writing this document OOPS! helps to automatically detect the following pitfalls from the catalogue (see Section 2): P2, P3, P4, P5, P6, P7, P8, P10, P11, P12, P13, P19, P20, P21, P22, P24, P25, P26, P27, P28, and P29.

Figure 1 shows OOPS! home page[3] where a user can enter an ontology to be analyzed via URL or RDF coding. It also presents a brief description of OOPS!.



OOPS! (OntOlogy Pitfalls Scanner!) helps you to detect some of the most common pitfalls appearing when developing ontologies.
To try it, enter a URI or paste an RDF/XML document into the text field above. A list of pitfalls and the elements of your ontology where they appear will be displayed.

Detecting common pitfalls in ontologies

Modelling ontologies has become one of the main topics of research within ontological engineering because of the difficulties it involves. Developers must tackle a wide range of difficulties and handicaps when modelling ontologies that can imply the appearance of anomalies or errors in ontologies. Therefore, it is important to evaluate the ontologies in order to detect those potential problems.

In this sense, OOPS! helps you to detect some of the most common pitfalls appearing within ontology developments. For example, OOPS! warns you when:

- The domain or range of a relationship is defined as the intersection of two or more classes. This warning could avoid reasoning problems in case those classes could not share instances.
- No naming convention is used in the identifiers of the ontology elements. In this case the maintainability, the accessibility and the clarity of the ontology could be improve.
- A cycle between two classes in the hierarchy is included in the ontology. This could avoid modelling and reasoning problems.
- And many other problems described in the catalogue.

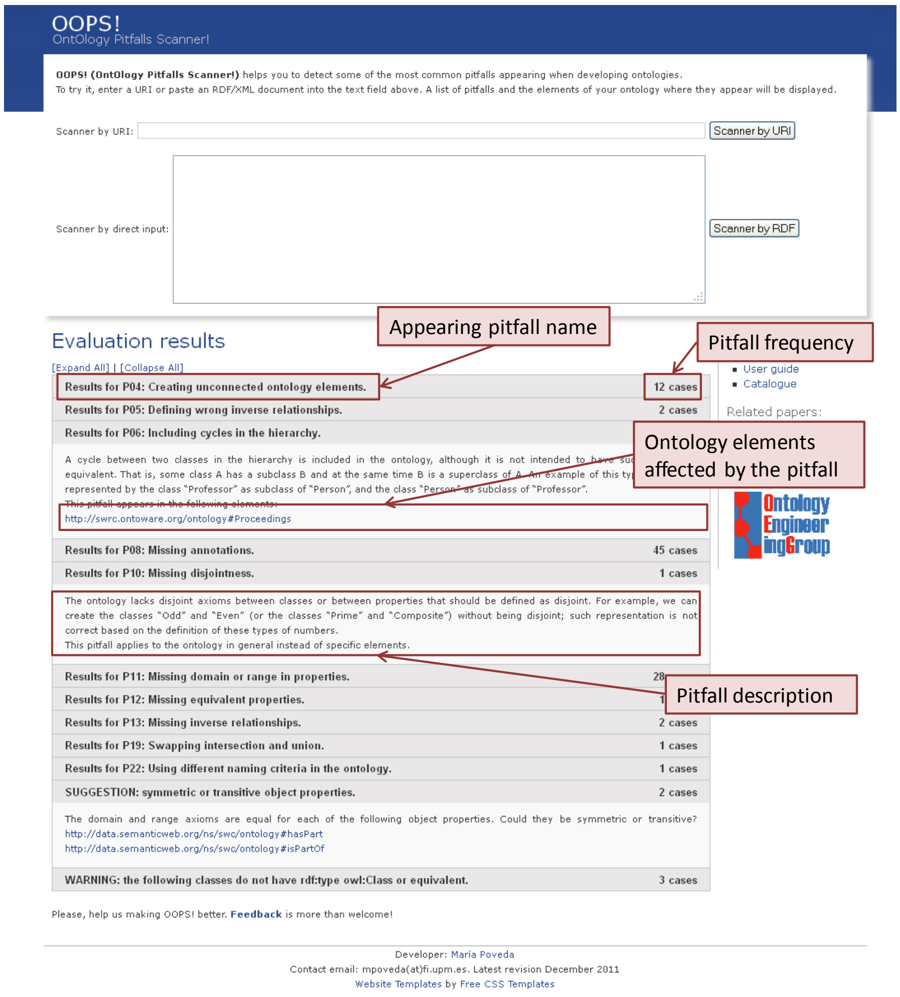
Please, help us making OOPS! better. **Feedback** is more than welcome!

Developer: María Poveda
Contact email: mpoveda(at)fi.upm.es. Latest revision December 2011
Website Templates by Free CSS Templates

Figure 1. OOPS! home page

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

After entering an ontology OOPS! generates, as it is shown in Figure 2, a new web page listing the appearing pitfalls in the ontology, how many times do they appear and which concrete ontology elements are affected. The same information is shown for warnings and suggestions proposed by OOPS! for some ontology elements.



OOPS!
OntOlogy Pitfalls Scanner!

OOPS! (OntOlogy Pitfalls Scanner!) helps you to detect some of the most common pitfalls appearing when developing ontologies.
To try it, enter a URI or paste an RDF/XML document into the text field above. A list of pitfalls and the elements of your ontology where they appear will be displayed.

Scanner by URI:

Scanner by direct input:

Evaluation results

[Expand All] | [Collapse All]

```
Results for P04: Creating unconnected ontology elements.                12 cases
Results for P05: Defining wrong inverse relationships.                   2 cases
Results for P06: Including cycles in the hierarchy.
```

A cycle between two classes in the hierarchy is included in the ontology, although it is not intended to have su equivalent. That is, some class A has a subclass B and at the same time B is a superclass of A. An example of this ty represented by the class "Professor" as subclass of "Person", and the class "Person" as subclass of "Professor".
This pitfall appears in the following elements:
http://swrc.ontoware.org/ontology#Proceedings

```
Results for P08: Missing annotations.                                   45 cases
Results for P10: Missing disjointness.                                   1 cases
```

The ontology lacks disjoint axioms between classes or between properties that should be defined as disjoint. For example, we can create the classes "Odd" and "Even" (or the classes "Prime" and "Composite") without being disjoint; such representation is not correct based on the definition of these types of numbers.
This pitfall applies to the ontology in general instead of specific elements.

```
Results for P11: Missing domain or range in properties.                 28
Results for P12: Missing equivalent properties.                          1
Results for P13: Missing inverse relationships.                          2 cases
Results for P19: Swapping intersection and union.                        1 cases
Results for P22: Using different naming criteria in the ontology.        1 cases
SUGGESTION: symmetric or transitive object properties.                   2 cases
```

The domain and range axioms are equal for each of the following object properties. Could they be symmetric or transitive?
http://data.semanticweb.org/ns/swc/ontology#hasPart
http://data.semanticweb.org/ns/swc/ontology#isPartOf

```
WARNING: the following classes do not have rdf:type owl:Class or equivalent.   3 cases
```

Please, help us making OOPS! better. **Feedback** is more than welcome!

Developer: María Poveda
Contact email: mpoveda(at)fi.upm.es. Latest revision December 2011
Website Templates by Free CSS Templates

Figure 2. Example of evaluation results web page generated by OOPS!

4 OOPS! architecture

In this section OOPS! underlying architecture is presented (see Figure 3) as well as some technical details. Basically, OOPS! is a web application based on JAVA EE[4],

[4] http://www.oracle.com/technetwork/java/javaee/overview/index.html

HTML[5], jQuery[6], JSP[7] and CSS[8] technologies. The web user interface consists on a simple view where the user enters the URL pointing to or the RDF document describing the ontology to be analyzed. Once the ontology is parsed using the JENA API[9] the model is scanned. During this phase, the ontology elements involved in potential errors are detected as well as warnings regarding RDF syntax and some modeling 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 each appearing pitfall.

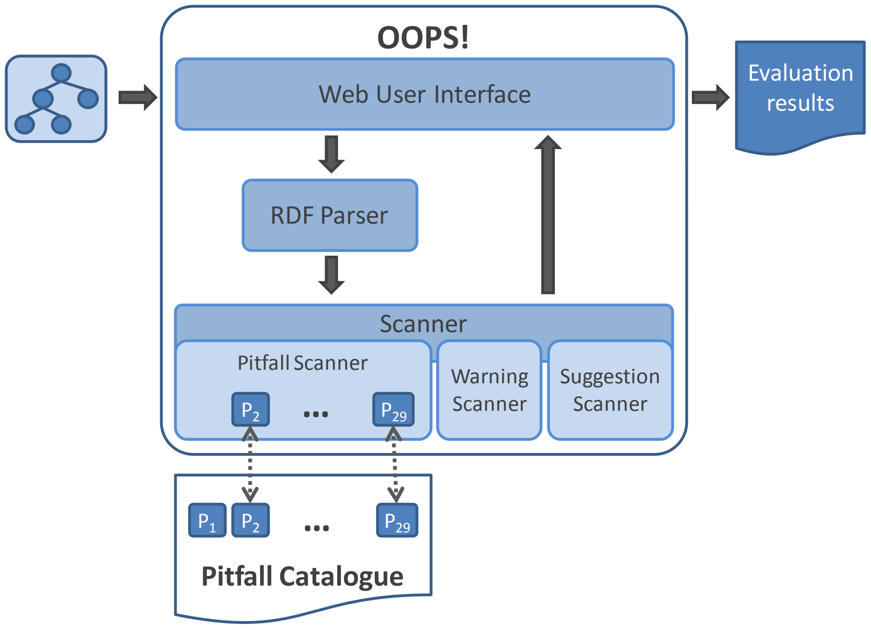


Figure 3. OOPS! architecture

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. Gómez-Pérez, A. *Ontology Evaluation*. Handbook on Ontologies. S. Staab and R. Studer Editors. Springer. International Handbooks on Information Systems. Pp: 251-274. 2004.

[5] http://www.w3.org/html/wg/
[6] http://jquery.com/
[7] http://www.oracle.com/technetwork/java/javaee/jsp/index.html
[8] http://www.w3.org/Style/CSS/
[9] http://jena.sourceforge.net/

4. 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.
5. 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). Proceedings of the Workshop on Ontology Quality - OntoQual 2010. ISBN: ISSN 1613-0073. CEUR Workshop Proceedings. Pages: 1-12. 15 October 2010. Lisbon, Portugal.
6. 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.
7. Suárez-Figueroa, M.C. *PhD Thesis: NeOn Methodology for Building Ontology Networks: Specification, Scheduling and Reuse*. Spain. Universidad Politécnica de Madrid. June 2010.
8. Suárez-Figueroa, M.C., Gómez-Pérez, A., Villazón-Terrazas, B. *How to write and use the Ontology Requirements Specification Document*. In Proceedings of ODBASE 2009. On the OTM 2009. LNCS 5871. Pp: 966-982. Vilamoura, Algarve-Portugal. November 2009.