OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation

María Poveda-Villalón, Ontology Engineering Group, Universidad Politécnica de Madrid, Madrid, Spain

Asunción Gómez-Pérez, Ontology Engineering Group, Universidad Politécnica de Madrid, Madrid, Spain

Mari Carmen Suárez-Figueroa, Ontology Engineering Group, Universidad Politécnica de Madrid, Madrid, Spain

ABSTRACT

This paper presents two contributions to the field of Ontology Evaluation. First, a live catalogue of pitfalls that extends previous works on modeling errors with new pitfalls resulting from an empirical analysis of over 693 ontologies. Such a catalogue classifies pitfalls according to the Structural, Functional and Usability-Profiling dimensions. For each pitfall, we incorporate the value of its importance level (critical, important and minor) and the number of ontologies where each pitfall has been detected. Second, OOPS! (OntOlogy Pitfall Scanner!), a tool for detecting pitfalls in ontologies and targeted at newcomers and domain experts unfamiliar with description logics and ontology implementation languages. The tool operates independently of any ontology development platform and is available online. The evaluation of the system is provided both through a survey of users' satisfaction and worldwide usage statistics. In addition, the system is also compared with existing ontology evaluation tools in terms of coverage of pitfalls detected.

Keywords: *Ontology, Ontology Evaluation, Ontology Quality, Ontology Validation, Pitfalls*

INTRODUCTION

The Linked Data (LD) effort has become a catalyst for the realization of the vision of the Semantic Web originally proposed by Berners-Lee et al. (2001). In this scenario, a large amount of data, annotated by means of ontologies, is shared on the Web. Such ontologies enrich the published data with semantics and help their integration. In other cases, ontologies are used to model data automatically extracted from web sources, which can be noisy and contain errors. Therefore, ontologies not only must be published according to LD principles[1], but they also must be accurate and of high quality from a knowledge representation perspective in order to avoid inconsistencies or undesired inferences.

The correct application of ontology development methodologies (e.g., METHONTOLOGY (Fernández-López et al., 1999), On-To-Knowledge (Staab et al., 2001), DILIGENT (Pinto, Tempich, & Staab, 2004), or the NeOn

DOI: 10.4018/ijswis.2014040102

Methodology (Suárez-Figueroa et al., 2012)) benefits the quality of the ontology being built. However, such a quality is not totally guaranteed because ontologists face a wide range of difficulties and handicaps when modeling ontologies (Aguado de Cea et al., 2008; Blomqvist, Gangemi, & Presutti, 2009; Rector et al., 2004), and this fact may cause the appearance of anomalies in ontologies. Therefore, in any ontology development project it is vital to perform the ontology evaluation activity since this activity checks the technical quality of an ontology against a frame of reference.

In the last decades a huge amount of research and work on ontology evaluation has been conducted. Some of these attempts define a generic quality evaluation framework (Duque-Ramos et al., 2011; Gangemi et al., 2006; Gómez-Pérez, 2004; Guarino, & Welty, 2009; Strasunskas, & Tomassen, 2008); others propose evaluating an ontology depending on its final (re)use (Suárez-Figueroa, 2010); some others propose quality models based on features, criteria, and metrics (Burton-Jones et al, 2005); whereas others present methods for pattern-based evaluation (Djedidi, & Aufaure, 2010; Presutti et al., 2008).

As a consequence of the emergence of new methods and techniques, a few tools have been proposed. These tools ease the ontology diagnosis by reducing the human intervention. This is the case of XD-Analyzer[2], a plug-in for NeOn Toolkit and Ontocheck[3] (Schober et al., 2012), a plug-in for Protégé. The former checks some structural and architectural ontology features, whereas the latter focuses on metadata aspects. Moki[4] (Pammer, 2010), a wiki-based ontology editor, also provides some evaluation features. Finally, Radon (Ji et al., 2009) is a NeOn Toolkit plug-in that detects and handles logical inconsistencies in ontologies.

This paper presents two main contributions. The first contribution consists of a live and on-line catalogue of pitfalls[5] that extends previous works on modeling errors (Allemang, & Hendler, 2011; Gómez-Pérez, 2004; Noy, & McGuinness, 2001; Rector et al., 2004) identified in the ontology engineering field including some persistent problems of accessibility emerging in the Linked Data field (Archer, Goedertier, & Loutas, 2012; Heath, & Bizer, 2011; Hogan et al., 2010). The second contribution, OOPS! (OntOlogy Pitfall Scanner!) represents a tool for diagnosing (semi-) automatically OWL[6] ontologies. This system aims to help ontology developers to evaluate ontologies and is focused on newcomers and those not familiar with description logics and ontology implementation languages. OOPS! operates independently of any ontology development platform and is available online at http://www.oeg-upm.net/oops. It should be noted here that the repair of the ontology is out of the scope of OOPS!.

In this paper we first present the catalogue of pitfalls, including a compendium of pitfalls extracted from the literature review and from the manual analysis of ontologies. A classification of such pitfalls according to the Structural, Functional and Usability-Profiling dimensions proposed in Gangemi et al. (2006) is also provided. Then, for each pitfall, we incorporate its value of importance level (critical, important, and minor) because not all the pitfalls are equally relevant and important. Next, we explain the internal architecture of OOPS! and describe the pitfalls detection methods used within the system. After that, an empirical analysis of the proposed catalogue carried out on 693 ontologies is presented. Then, we present the evaluation of the system based both on a survey of users' satisfaction and on evidence of the real use of the tool worldwide. After that, we review related works about ontology evaluation tools. Finally we draw the conclusions and provide future lines of work.

COMMON PITFALLS IN ONTOLOGY DEVELOPMENT

One of the most common approaches for evaluating ontologies is to have a checklist of typical errors that other developers have made before. Thus the developer checks the ontology being built against such a list, detects the

pitfalls, and corrects them. Our approach does not pretend to create another checklist but to reuse existing works where modeling problems have already been identified and to extend them by incorporating new pitfalls obtained through an empirical evaluation of ontologies already existing.

Catalogue of Common Pitfalls

As our long-term goal is to create and maintain a live and on-line pitfall catalogue, we have followed the process sketched in Figure 1. We started by manually analyzing ontologies and reviewing literature about ontology evaluation and Linked Data (LD).

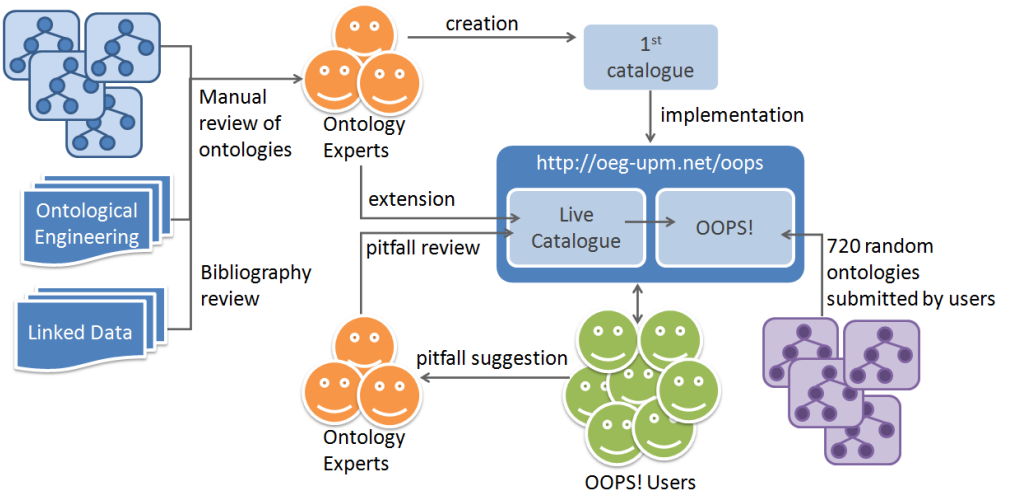
Regarding works on ontology evaluation, we reviewed, reused, and included in the pitfall catalogue outcomes from (Rector et al., 2004), in which Rector et al. describe a set of common errors made by developers during the ontology modeling activity; from Gómez-Pérez (2004), in which Gómez-Pérez provides a classification of errors identified during the evaluation of consistency, completeness, and conciseness of ontology taxonomies; and from Noy, and McGuinness (2001), where Noy and McGuinness present a methodology for creating ontologies and point out some common errors and how to avoid them. We have also reused and adapted to the ontology domain some research from the LD area: the main guidelines for LD publication and consumption (Heath, & Bizer, 2011); the problems identified in Hogan et al. (2010) for accessing RDF[7] on the Web; and the guidelines for creating persistent URIs included in Archer, Goedertier, and Loutas (2012).

The catalogue does not pretend to be an exhaustive, rigid and fixed checklist. Besides, in order to keep such a catalogue in continuous evolution we continue working with the manual evaluation of ontologies and aim to discover new pitfalls. We would welcome that OOPS! users and ontology experts propose new pitfalls to introduce them in the catalogue.

The current version of the catalogue[8] consists of a list of 40 pitfalls as well as their descriptions. In each pitfall we include provenance information if the pitfall being described was proposed in a previous work. The list includes the following pitfalls:

- **P01. 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

Figure 1. Creation of the pitfall catalogue and maintenance process



to represent both the artistic discipline and the place in which a play is performed.

- **P02. Creating Synonyms as Classes:** Several classes whose identifiers are synonyms are created and defined as equivalent. For example, the classes "Waterfall" and "Cascade" are defined as equivalents. This pitfall is related to the guidelines presented in Noy, and McGuinness (2001), which explain that synonyms for the same concept do not represent different classes.
- **P03. Creating the Relationship "is" Instead of Using "rdfs:subClassOf", "rdf:type" or "owl:sameAs":** 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 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 Noy, and McGuinness (2001).
- **P04. 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.
- **P05. Defining Wrong Inverse Relationships:** Two relationships are defined as inverse relations when they are not necessarily inverse. An example of this type of pitfall is to define "isSoldIn" and "isBoughtIn" as inverse relationships.
- **P06. Including Cycles in the Hierarchy (Gómez-Pérez, 2004; Noy, & McGuinness, 2001):** A cycle between two classes in the hierarchy is included in the ontology even though the ontology 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".
- **P07. Merging Different Concepts in the Same Class:** A class whose identifier refers to two or more different concepts is created. An example of this type of pitfall is the creation of the class "StyleAndPeriod".
- **P08. Missing Annotations:** Ontology terms lack annotations properties such as *rdfs:label* or *rdfs:comment*. An example of this type of pitfall is to create a class and to fail to provide human readable annotations attached to such a class.
- **P09. Missing Basic Information:** Some of the information needed is not included in the ontology. This pitfall may be related to the requirements in the ontology requirements specification document (ORSD) not covered by the ontology, or to knowledge that can be added to the ontology to make it more complete. An example of this type of pitfall is to create the relationship "startsIn" in order to represent that the routes have a starting point in a particular location and to miss the relationship "endsIn" in order to represent that a route has an end point.
- **P10. Missing Disjointness (Gómez-Pérez, 2004; Noy, & McGuinness, 2001; Rector et al., 2004):** 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 incomplete with regard to 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. An example of this type of pitfall is to create the relationship "hasWritten", with no domain nor range specification, 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 that appears when defining the ranges and domains described in Rector et al. (2004).

- **P12. Missing Equivalent Properties:** When an ontology is imported into another, classes duplicated in both ontologies are normally defined as equivalent classes. However, the ontology developer misses the definition of equivalent properties in the cases of duplicated relationships and attributes. An example of this type of pitfalls is to fail to define the relations "hasMember" and "has-Member" as equivalent.
- **P13. Missing Inverse Relationships:** This pitfall appears when any relationship (except for the symmetric ones) does not have an inverse relationship defined within the ontology. For example, the case in which the ontology developer omits the inverse definition between the relations "hasLanguageCode" and "isCodeOf".
- **P14. Misusing "owl:allValuesFrom" (Rector et al., 2004):** This pitfall can appear in two different ways. Firstly, when the universal restriction ("allValuesFrom") is used as the default qualifier instead of the existential restriction ("someValuesFrom"). Secondly, when "allValuesFrom" is included 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' thus closing the possibility of adding "Ink" as an element used in the writing.
- **P15. Misusing "not some" and "some not" (Rector et al., 2004):** The pitfall here is to confuse the representation of "some not" with "not some". An example of this type of pitfall is to define a vegetarian pizza as any pizza which has both some topping which is not meat and some topping which is not fish. This example is explained in more detail in Rector et al. (2004).
- **P16. Misusing Primitive and Defined Classes (Rector et al., 2004):** This pitfall implies failing 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 (Rector et al., 2004). This pitfall implies that the developer does not understand the open world assumption. An example of this pitfall is to create the primitive class 'CheesyPizza ⊏ Pizza ⨅ ∃hasTopping. Cheese' instead of creating it as a defined class in the following way: 'CheesyPizza ≡ Pizza ⨅ ∃hasTopping.Cheese'. This example is explained in more detail in Rector et al. (2004).
- **P17. Specializing a Hierarchy Exceedingly[9]:** The hierarchy in the ontology is specialized in such a way that the final leaves cannot have instances since they are actually instances and should have been created as such instead of as classes. Authors in Noy, and McGuinness (2001) provide guidelines for distinguishing between a class and an instance when modeling hierarchies. An example of this type of pitfall is to create the classes "Madrid", "Barcelona" and "Sevilla", among others, as subclasses of "Place".
- **P18. Specifying the Domain or the Range Exceedingly (Noy, & McGuinness, 2001; Rector et al., 2004):** This pitfall means failing to find a domain or a range 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 the class "Country" or a more general concept such as "GeopoliticalObject" to have an official language.
- **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 those classes. This pitfall is related both to the common error that appears when defining ranges and domains described in Rector et al. (2004) and to the

guidelines for defining these elements provided in Noy, and McGuinness (2001). An example of this type of pitfall is to create the relationship "takesPlaceIn" with one range declaration for the class "City" and other range declaration for the class "Nation", as this implementation represents the intersection of both ranges instead of the union.

- **P20. Misusing Ontology Annotations:** The contents of some annotation properties are swapped or misused. An example of this type of pitfall is to include in the *rdfs:label* annotation of the class "Crossroads" the following sentence 'the place of intersection of two or more roads'; and to include in the *rdfs:comment* annotation the word 'Crossroads'.
- **P21. Using a Miscellaneous Class:** This means creating in a hierarchy a class containing 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. An example of this type of pitfall is to create the class "HydrographicalResource", and the subclasses "Stream" and "Waterfall", among others, and also the subclass "OtherRiverElement".
- **P22. Using Different Naming Criteria in the Ontology:** Ontology elements are not named following the same convention within the whole ontology. Some notions about naming conventions are provided in Noy, and McGuinness (2001). For example, this pitfall appears when a class identifier starts with upper case, e.g. "Ingredient", whereas its subclass identifiers start with lower case, e.g. "flour" and "milk".
- **P23. Using Incorrectly Ontology Elements:** An ontology element (class, relationship or attribute) is used to model a part of the ontology that should be modeled with a different element. A particular case of this pitfall regarding the misuse of classes and property values is addressed in Noy, and McGuinness (2001). An example of this type of pitfall is to create the relationship "isEcological" between an instance of "Car" and the instances "Yes" or "No", instead of creating an attribute "isEcological" whose range is Boolean.
- **P24. Using Recursive Definition:** An ontology element is used in its own definition. An example of this type of pitfall is 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:** A relationship is defined as inverse of itself. 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:** A relationship is defined as "owl:SymmetricProperty", and such a relationship is defined as inverse of another relationship. For example, to create for the symmetric relationship "farFrom" an inverse relationship, e.g. itself, "farFrom".
- **P27. Defining Wrong Equivalent Relationships:** Two relationships are defined as equivalent relations when they are not necessarily equivalent. An example of this type of pitfalls is to mix up common relationships that could hold between several types of entities, as "hasPart" defined in one ontology between human body parts and the relation "hasPart" defined in another ontology between research plans and research projects.
- **P28. Defining Wrong Symmetric Relationships:** A relationship is defined as symmetric when the relationship is not necessarily symmetric. This situation can appear because the domain and range are too specific; 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 Wrong Transitive Relationships:** A relationship is defined as transitive when the relationship is not necessarily transitive. An example of this type of pitfall is to create the relationship "participatesIn", whose domain is the union of the concepts "Team" and "Individual" and whose range is the concept "Event", and defining the relationship as transitive.
- **P30. Missing Equivalent Classes:** When an ontology is imported into another, classes with the same conceptual meaning that are duplicated in both ontologies should be defined as equivalent classes in order to benefit the interoperability between both ontologies. However, the ontology developer may miss the definition of equivalent classes in the cases of duplicated concepts. An example of this pitfall is to fail to define the classes 'Trainer' (class in an imported ontology) and 'Coach' (class in the ontology about sports being developed) as equivalent classes.
- **P31. Defining Wrong Equivalent Classes:** Two classes are defined as equivalent when they are not necessarily equivalent. For example, defining "Car" as equivalent to "Vehicle".
- **P32. Several Classes with the Same Label:** Two or more classes have the same content in the *rdfs:label* annotation. For example, to link the label "Theatre" both with the building and the literary discipline, adding no more labels to them.
- **P33. Creating a Property Chain with Just One Property:** There is a property chain that includes only one property in the antecedent part. For example, to create the following property chain: isInChargeOf -> supervises.
- **P34. Untyped Class (Hogan et al., 2010):** A resource is used as a class without having been declared as a Class. An example of this type of pitfall is to create individuals of the class "Person" and to omit that "Person" is a class.
- **P35. Untyped Property (Hogan et al., 2010):** A resource is used as a property without having been declared as a rdf:Property or as some subclass of it. An example of this type of pitfall is to link individual by the relation "hasPart" and to omit that "hasPart" is an object property.
- **P36. URI Contains File Extension (Archer, Goedertier, & Loutas, 2012):** This involves including file extensions as ".owl", ".rdf", ".ttl", ".n3" and ".rdfxml" in an ontology URI. An example of this pitfall is to define an ontology uri as "http://www.biopax.org/release/biopax-level3.owl" containing the extension ".owl" related to the technology used.
- **P37. Ontology Not Available:** This involves omitting to provide online description or documentation of the ontology when looking up its URI. An example of this pitfall could be the following case: "Ontology Security (ontosec)" (URI: http://www.semanticweb.org/ontologies/2008/11/OntologySecurity.owl) which is not available online as RDF nor as HTML (at the moment of carrying out this work).
- **P38. No OWL Ontology Declaration:** This means failing to declare the *owl:Ontology* tag where the ontology metadata should be provided. An example of this pitfall could be found at the "Creative Commons Rights Expression Language (cc)" ontology (URI: http://creativecommons.org/ns*)* that does not have any *owl:Ontology* declaration in its RDF file even though it has other OWL elements used as, for example, owl:equivalentProperty (at the moment of carrying out this work).
- **P39. Ambiguous Namespace:** This means failing to define both the ontology URI and the *xml:base* namespace. An example of this pitfall could be found at "Basic Access Control ontology (acl)" (URI: http://www.w3.org/ns/auth/acl) that has no owl:Ontology tag nor xml:base definition.
- **P40. Namespace Hijacking (Heath, & Bizer, 2011):** This means reusing or referring to terms from other namespaces not actually defined in such namespace. This pitfall is related to the Linked Data publishing guidelines provided in Heath, and Bizer (2011): "Only define new terms in a namespace

that you control." An example of this pitfall is to use "http://www.w3.org/2000/01/rdf-schema#Property" that is not defined in the rdf namespace (http://www.w3.org/2000/01/rdf-schema#) instead of using "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property", that is actually defined in the rdfs namespace (http://www.w3.org/1999/02/22-rdf-syntax-ns#).

Pitfalls Classification

Since the list of pitfalls presented refers to different ontology perspectives, it is advisable to classify them according to some evaluation criteria. Users with an interest in a given aspect of ontology evaluation could easily identify the group of pitfalls in which they might be interested. For this reason, we have classified pitfalls according to the dimensions defined in Gangemi et al. (2006), namely: structural, functional and usability-profiling. Even though these dimensions are enough to classify all the pitfalls in the catalogue, a more fine-grained classification is provided to deal with specific aspects that following and extend the approach described in Poveda-Villalón, Suárez-Figueroa, and Gómez-Pérez (2010). Such classification is as follows:

- **Structural Dimension (Gangemi et al., 2006):** It is focused on syntax and formal semantics. For this dimension we consider the following aspects:
  - *Modeling Decisions* (Poveda-Villalón, Suárez-Figueroa, & Gómez-Pérez, 2010): This aspect involves evaluating whether developers use the primitives provided by ontology implementation languages in a correct way, and if there are modeling decisions that could be improved.
  - *Real World Modeling or Common Sense* (Poveda-Villalón, Suárez-Figueroa, & Gómez-Pérez, 2010): This aspect deals with the knowledge that domain experts expect to appear in the ontology, but is not represented.
  - *No Inference* (Poveda-Villalón, Suárez-Figueroa, & Gómez-Pérez, 2010): This aspect refers to checking whether desirable or expected knowledge could actually be inferred from the given ontology.
  - *Wrong Inference* (Poveda-Villalón, Suárez-Figueroa, & Gómez-Pérez, 2010): This aspect refers to the evaluation of the inference of erroneous or invalid knowledge.
  - *Ontology Language:* This aspect refers to checking whether the ontology is compliant both with the ontology language specification and with the syntax in which the ontology is formalized.
- **Functional Dimension (Gangemi et al., 2006):** This is related to the intended use of a given ontology; thus the focus is on the ontology conceptualization. The following aspects are taken into account within this dimension:
  - *Requirement Completeness* (Poveda-Villalón, Suárez-Figueroa, & Gómez-Pérez, 2010): This aspect deals with the coverage of the requirements specified in the ORSD by the ontology.
  - *Application Context:* This aspect refers to the adequacy of the ontology for a given application or use case.
- **Usability-Profiling Dimension (Gangemi et al., 2006):** It refers to the communication context of an ontology. For this dimension we contemplate the following aspects:
  - *Ontology Understanding* (Poveda-Villalón, Suárez-Figueroa, & Gómez-Pérez, 2010): This aspect involves evaluating any kind of information that can help the user to understand the ontology.
  - *Ontology Clarity* (Poveda-Villalón, Suárez-Figueroa, & Gómez-Pérez, 2010): This aspect refers to the properties of ontology elements of being easily recognizable and understood by the user.

Figure 2 represents such classification where each pitfall is classified according to at least one of the abovementioned aspects. Figure 2 also shows the importance level of each pitfall both by attaching a number between brackets to each pitfall title and by using different colors; thus "critical" pitfalls are written in black followed by "(1)", "important" pitfalls are in blue followed by "(2)" and "minor" pitfalls are in brown followed by "(3)".

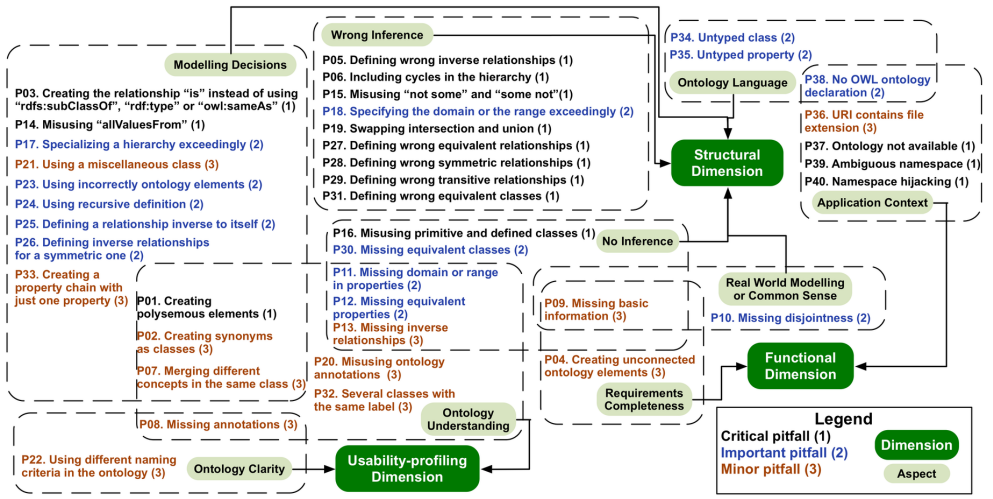
Extension with Pitfalls Importance Levels

It is obvious that not all the pitfalls are equally important; their impact in the ontology will depend on multiple factors. For this reason, the pitfall catalogue has been extended with information about how critical the pitfalls are. We have identified three levels:

- **Critical (1):** It is crucial to correct the pitfall. Otherwise, it could affect the ontology consistency, reasoning and applicability, among others. For example, the consequences of "P19. Swapping intersection and union" could lead to logical inconsistences in the ontology, which represents a critical error when reasoning over the populated ontology.
- **Important (2):** Though not critical for ontology function, it is important to correct this type of pitfall. For example, the logical consequences of "P25. Defining a relationship inverse to itself" are the same as if such relationship were defined as symmetric. However, the latter option, that is, using the ontology implementation language constructors for the purpose they were conceived, is a sign of good modeling and understanding of the underlying semantics.
- **Minor (3):** It does not represent a problem. However, correcting it makes the ontology better organized and user friendly. For example, the pitfall "P22. Using different naming criteria in the ontology" is about the appearance of the ontology and does not compromise the proper ontology functioning.

These levels do not have clear boundaries in the sense that a particular pitfall in a level could be debatable depending on the modeling styles, ontology requirements, and context of use by an ontology application. For example, in

Figure 2. Pitfall classification



this work we consider an important pitfall not to define domains and ranges for the properties, which is arguable, of course. In some developments, it could be considered a pitfall exactly the opposite (that is, specifying domains and ranges), as developers might be interested in increasing the interoperability of the model obtained instead of its explicit semantics or expressivity. In such a case, it would be enough if the evaluators define the fact of defining domains and ranges as a pitfall instead of doing it as we propose here. In this way, we provide a starting point for ontology evaluation that could be adapted to users' particular requirements.

In other cases, how critical a pitfall is depends on the context of use; for example, in a LD development project (Heath, & Bizer, 2011), an ontology should be published according to the Linked Data rules and principles. In this scenario, the pitfalls "P37. Ontology not available on the Web", "P39. Ambiguous namespace", and "P40. Namespace hijacking" are crucial while they might not be important in the context of an isolated application where the ontology is not designed for sharing. Another pitfall related to LD context is "P36. URI contains file extension". In this case, it may be consider a minor pitfall as it does not affect the correct functioning of the ontology.

At the moment of including the importance levels in the catalogue, 35 out of the 40 pitfalls were already defined and published. In order to attach importance levels to the pitfalls, a study was carried out in which the users had to fill in a questionnaire providing the following information:

- **Level of Confidence:** How confident (s)he felt in the ontology evaluation or ontology modeling domains.
- **Importance Level of Each Pitfall:** There was one question per pitfall (from P01 to P35) where the user had to select the importance level of the given pitfall. The possible values were "Critical", "Important" and "Minor" (see above).
- **Which Pitfalls are Not Important:** A list with all the pitfalls was provided and the users were asked to indicate which pitfall would never represent a problem (not pitfalls that could be a problem only in some cases) for them.
- **Other Comments:** A free text box for providing any comment or suggestions.

Researchers, mainly experts on ontology modeling or evaluation, within the semantic web community[10] and OOPS! users were invited to fill in the questionnaire. We received 55 responses. We have made the questionnaire available on-line at http://goo.gl/SEddMN in order to allow the community to continue with the assessment of the level of importance of the pitfalls. On the other hand, to assign importance levels to pitfalls according to the data gathered through the survey, we have first assigned weight to each response (3 for critical[11], 2 for important, and 1 for minor) and to each expertise level (3 for experts, 2 for medium confidence, and 1 for low confidence). For those pitfalls selected as "not important", we have assigned the weight 0 in the corresponding response. The data generated from the survey responses and the ranking calculations are available at the URL: http://goo.gl/0IkbS2

Then we have ranked the pitfalls according to the well-known "weighted sum" technique and obtained the ranking shown in the first column from the left in Table 1.

Once the pitfalls are ranked, an interval should be defined in order to split the given ranking into 3 parts, one for each importance level. To do this, we have used a method based on the range of the weight values. More precisely, the range (highest weight – lowest weight) is divided into 3. Concretely, the range of the weighted sum ranking is 0.0193 (0.0379 – 0.0186). The division of the range among 3 gives us an interval of 0.0064.

Finally, the range of the ranking is split into 3, resulting in the following intervals:

- **Minor:** From 0.0186 to 0.0250 (0.0186 + 0.0064)
- **Important:** From 0.0250 to 0.0314 (0.0250 +0.0064)

Table 1. Pitfalls (from P01 to P35) ranked according to the (a) weighted sum, (b) lexicographic order and (c) centroid function techniques

	(a) Weighted sum		(b) Lexicographic order	(c) Centroid function		
	Order	**Weight**	**Order**	**Order**	**Weight**	
Critical (1)	P06. Including cycles in the hierarchy	0.0379	P06	P06	0.0366	Critical (1)
	P19. Swapping intersection and union	0.0375	P19	P19	0.0359	
	P01. Creating polysemous elements	0.0367	P03	P29	0.0351	
	P03. Creating the relationship “is” instead of using ‘’rdfs:subClassOf’’, ‘’rdf:type’’ or ‘’owl:sameAs”	0.0364	P01	P01	0.0346	
	P29. Defining wrong transitive relationships	0.0348	P29	P03	0.0346	
	P28. Defining wrong symmetric relationships	0.0344	P14	P31	0.0343	
	P31. Defining wrong equivalent classes	0.0343	P31	P15	0.0336	
	P05. Defining wrong inverse relationships	0.0342	P16	P14	0.0336	
	P14. Misusing ‘’owl:allValuesFrom’’	0.0341	P15	P28	0.0335	
	P27. Defining wrong equivalent relationships	0.0340	P27	P16	0.0333	
	P15. Misusing “not some” and “some not”	0.0335	P28	P27	0.0330	
	P16. Misusing primitive and defined classes	0.0335	P05	P05	0.0318	
Important (2)	P23. Using incorrectly ontology elements	0.0303	P24	P24	0.0312	
	P24. Using recursive definition	0.0303	P12	P23	0.0303	Important (2)
	P12. Missing equivalent properties	0.0301	P10	P12	0.0303	
	P34. Untyped class	0.0284	P23	P10	0.0287	
	P10. Missing disjointness	0.0283	P34	P34	0.0286	
	P35. Untyped property	0.0281	P35	P30	0.0283	
	P25. Defining a relationship inverse to itself	0.0279	P11	P35	0.0283	
	P30. Missing equivalent classes	0.0279	P25	P11	0.0275	
	P18. Specifying the domain or range exceedingly	0.0272	P26	P25	0.0273	
	P26. Defining inverse relationships for a symmetric one	0.0272	P18	P26	0.0270	
	P17. Specializing a hierarchy exceedingly	0.0267	P17	P18	0.0267	
	P11. Missing domain or range in properties	0.0252	P30	P17	0.0261	

continued on following page

Table 1. Continued

	(a) Weighted sum		(b) Lexicographic order	(c) Centroid function		
	Order	**Weight**	**Order**	**Order**	**Weight**	
Minor (3)	P04. Creating unconnected ontology elements	0.0248	P04	P07	0.0253	Minor (3)
	P09. Missing basic information	0.0245	P07	P04	0.0253	
	P33. Creating a property chain with just one property	0.0240	P02	P09	0.0245	
	P02. Creating synonyms as classes	0.0239	P09	P33	0.0237	
	P07. Merging different concepts in the same class	0.0234	P33	P02	0.0236	
	P21. Using a miscellaneous class	0.0222	P21	P32	0.0226	
	P32. Several classes with the same label	0.0219	P13	P21	0.0226	
	P13. Missing inverse relationships	0.0201	P32	P13	0.0215	
	P22. Using different naming criteria in the ontology	0.0189	P20	P20	0.0206	
	P20. Misusing ontology annotations	0.0187	P08	P08	0.0205	
	P08. Missing annotations	0.0186	P22	P22	0.0200	

- **Critical:** From 0.0314 to 0.0379

In order to demonstrate that the ranking method selected is robust, we compared it with two other ranking methods, namely, the "lexicographic order" (Miettinen, 1999) and the "centroid function" (Barron, & Barrett, 1996). The rankings obtained for these methods are shown in Table 1, more precisely, in the second and third columns from the left respectively. For the case of the "centroid function" we have also calculated the intervals for the "Critical", "Important", and "Minor" categories in the same manner as explained for the "weighted sum". As the "lexicographic order" does not involve weights or ranges, it does not make sense to split the range in this fashion. More precisely, the lexicographic order is calculated as follows: first, the pitfalls are ordered according to the votes that the value "critical (3)" attained. The more votes attained, the higher the pitfall is placed in the ranking. For example, the P06 is first with 46 votes[12]. When two or more pitfalls have the same number of votes in this category, the information about the next importance levels is used to break the tie. For example, P29 and P14 have 37 votes for the value "critical", so the votes for "important (2)" are used, that is, the P14 is placed first with 12 votes, and P14 is next with 9 votes.

Once the rankings were computed, we analyzed the similitudes and differences between them. That is, given two rankings we measure how similar the orders established for the list of pitfalls are. To do so, we calculated the Kendall coefficient (Winkler & Hays, 1985), being the values obtained for each pair of rankings the following[13]:

- **Weighted Sum – Lexicographic Order:** 0.882352941
- **Weighted Sum – Centroid Function:** 0.905882353
- **Lexicographic Order – Centroid Function:** 0.929411765

We can observe that the three values are very high; this fact means that the rankings are very similar and proves that the decision of choosing the weighted sum does not affect significantly the final classification. In fact, there is only one pitfall, "P24. Using recursive

definition" that has been attached to different importance levels according to the weighted sum method (classified as "important") and to the centroid function method (classified as "critical").

When a new pitfall is inserted in the catalogue, an importance level has to be assigned to it. This importance level is decided in conjunction with the developers of OOPS!, experienced ontological engineers, and the users (if any) proposing the given pitfall. For the pitfalls P36 to P40, four experts in ontological engineering and vocabulary publication have defined the pitfalls and assigned their importance levels. As a result, the importance levels shown in Table 2 have been attached to each pitfall.

Taking into account the importance levels extracted from the survey and those levels assigned by ontology experts, we have created a final classification of pitfalls as shown in Figure 3[14].

OOPS! (ONTOLOGY PITFALL SCANNER!)

OOPS! is a web-based tool for diagnosing potential problems in ontologies that could lead to modeling errors. This tool is intended to help ontology developers, mainly newcomers, during the ontology validation activity (Suárez-Figueroa, Aguado-de-Cea, & Gómez-Pérez, 2013). Currently, OOPS! provides mechanisms to (semi-)automatically diagnose 32 pitfalls of the 40 described in the pitfall catalogue as Figure 3 shows.

This section is divided into two parts: the first subsection explains the internal architecture of OOPS!; and the second subsection describes the detection methods used within the system in order to spot pitfalls in the ontology analyzed.

OOPS! Architecture

Figure 4 presents the underlying architecture of OOPS!. OOPS! is a web application based on Java EE[15], HTML[16], jQuery[17], JSP[18] and CSS[19] technologies. In order to produce a list of evaluation results, OOPS! takes as input both the pitfall catalogue and an ontology.

The user interface consists of a webpage, in which the user enters either the ontology URI or its OWL code, which describes the ontology to be analyzed. Once the ontology is parsed using the Jena API[20], the "Pitfall Scanner" module inspects the declared ontology[21] looking for pitfalls among those available in the catalogue. More precisely, the 32 pitfalls implemented are those that can be detected (semi-) automatically with the information provided by the ontology OWL code (T-box). Those pitfalls that require an external reference framework (e.g., an ontology requirement document, an A-box or corpora, and/or domain knowledge) or human intervention are not yet automated. During this scanning phase, the ontology elements prone to potential errors are detected, whereas some modeling suggestions are generated by the "Suggestion Scanner" module. Finally, the evaluation results are displayed in the web user interface, which shows the list of pitfalls detected, if any, and the ontology elements affected, as well as explanations describing the findings (Figure 5). The web interface allows not only analyzing all the automated pitfalls, but also choosing specific pitfalls or predefined groups according to the

Table 2. Importance levels for pitfalls assigned by experts

	Pitfalls
Critical (1)	• P37. Ontology not available • P39. Ambiguous namespace • P40. Namespace hijacking
Important (2)	• P38. No OWL ontology declaration
Minor (3)	• P36. URI contains file extension

Figure 3. Classification of pitfalls by level of importance

CRITICAL (1)	
P01. Creating polysemous elements	
P03. Creating the relationship "is" instead of using "rdfs:subClassOf", "rdf:type" or "owl:sameAs"	
P05. Defining wrong inverse relationships	
P06. Including cycles in the hierarchy	
P14. Misusing "owl:allValuesFrom"	
P15. Misusing "not some" and "some not"	
P16. Misusing primitive and defined classes	
P19. Swapping intersection and union	
P27. Defining wrong equivalent relationships	
P28. Defining wrong symmetric relationships	
P29. Defining wrong transitive relationships	
P31. Defining wrong equivalent classes	
P37. Ontology not available	
P39. Ambiguous namespace	
P40. Namespace hijacking	
IMPORTANT (2)	**MINOR (3)**
P10. Missing disjointness	**P02. Creating synonyms as classes**
P11. Missing domain or range in properties	**P04. Creating unconnected ontology elements**
P12. Missing equivalent properties	**P07. Merging different concepts in the same class**
P17. Specializing a hierarchy exceedingly	**P08. Missing annotations**
P18. Specifying the domain or range exceedingly	P09. Missing basic information
P23. Using incorrectly ontology elements	**P13. Missing inverse relationships**
P24. Using recursive definition	**P20. Misusing ontology annotations**
P25. Defining a relationship inverse to itself	**P21. Using a miscellaneous class**
P26. Defining inverse relationships for a symmetric one	**P22. Using different naming criteria in the ontology**
P30. Missing equivalent classes	**P32. Several classes with the same label**
P34. Untyped class	**P33. Creating a property chain with just one property**
P35. Untyped property	**P36. URI contains file extension**
P38. No OWL ontology declaration	

pitfall classification presented in this paper. This "Advanced evaluation" feature is linked from the homepage and available at index http://www.oeg-upm.net/oops/advanced.jsp.

Furthermore, to allow other programs and applications to use OOPS! pitfall detection methods, we have developed a web service[22].

Next subsection describes the different approaches used to implement the methods for detecting pitfalls.

Pitfall Detection Methods

The pitfall catalogue covers many different aspects of ontologies, such as their internal structure, their associated or embedded human-readable documentation, or their availability on the Web. As a consequence, the detection methods implemented to detect pitfalls make use of different techniques and technologies for diagnosing them. More precisely, the detection

Figure 4. OOPS! architecture

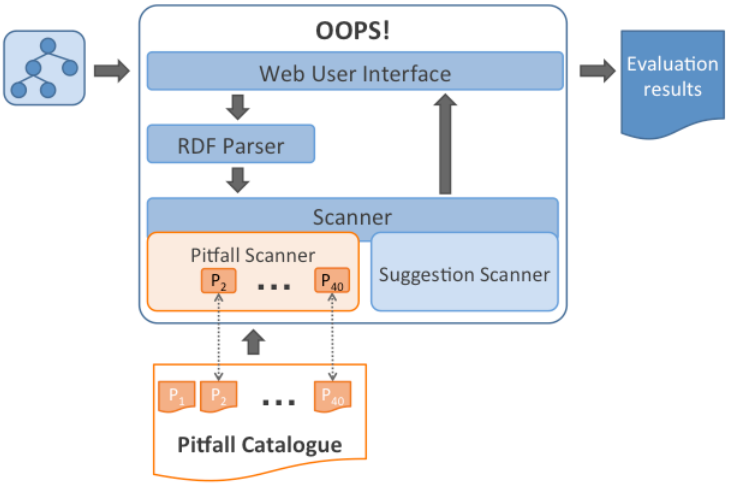
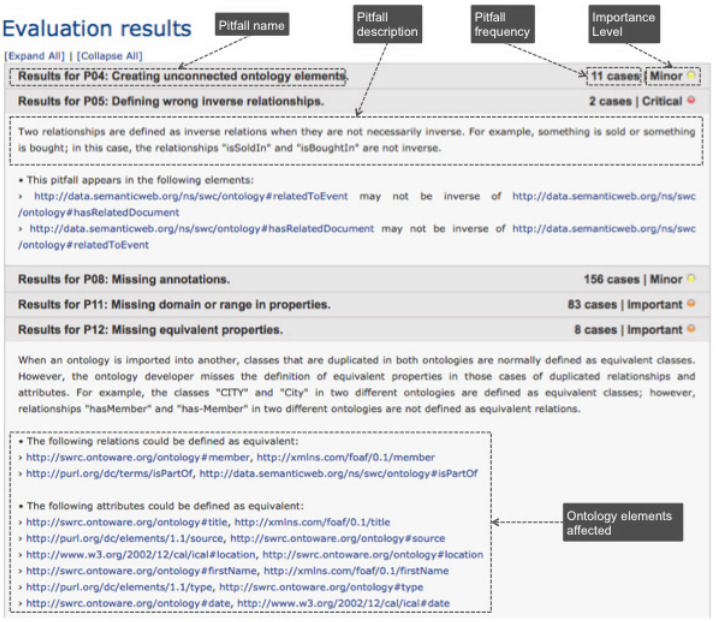


Figure 5. OOPS! response example



methods used within OOPS! are based on one (or more) of the following approaches:

- **Structural Pattern Matching:** The detection methods based on patterns analyze the internal structure of the ontology, seeking specific parts of the model. In these cases, a pitfall is diagnosed when a given structural pattern is spotted. Of the 32 pitfalls, 24 have been implemented using structural patterns. A number of these structural patterns are shown in Figure 6 and Figure 7. In such figures, classes are represented by rectangles, properties by plain arrows, individuals by ellipses, and OWL and RDFS primitives by dotted arrows. Properties can also be represented by diamonds including property characteristics (e.g. transitive). The patterns can also include statements following the OWL functional syntax, mainly to indicate that the pattern checks the lack of such information. It should be noted that some pitfalls are detected by different patterns, for example, "P19. Swapping intersection and union"; in those cases the pitfall is detected when at least one of the patterns is identified in the ontology.
- **Lexical Content Analysis:** The detection methods based on the analysis of lexical entities make use of the content of annotations (e.g., *rdfs:label* or *rdfs:comment*) and identifiers (the ID part of the element URI) for detecting pitfalls. These methods are used in 9 of the 32 implemented pitfalls. For the pitfall "P22. Using different naming criteria in the ontology", the identifiers of the ontology elements are analyzed to check whether all of them use the same naming convection for example, if all the identifiers are formed according to the CamelCase rules.
- **Specific Characteristic Search:** Five detection methods have been automated by checking general characteristics of the ontology not related to the internal structure of the ontology or to the content of the lexical entities. These characteristics could be related, for example, to the name given to the ontology as in the pitfall "P36. URI contains file extension", which is detected when the ontology URI refers to the technology or ontology language used during its development as RDF or OWL. Detailed technical information about detection methods for seeking a specific characteristic can be found in Poveda-Villalón et al. (2013).

In addition, some pitfalls can appear several times in the same ontology while others may appear at most once, since they affect the whole ontology instead of its different elements (classes, properties and axioms, among others).

Figure 8 shows the type of technique(s) used for detecting each pitfall and its cardinality, that is, how many times such a pitfall could be spotted in a given ontology. For example, we can observe that the pitfall "P11. Missing domain or range in properties" is detected by seeking a given pattern and that it could appear more than once, or more precisely, it could appear as many times as relations are defined in the ontology.

There are cases where a detection method uses more than one technique as indicated in Figure 8, with the rectangles located between two cells. For example, to detect "P30. Missing equivalent classes", OOPS! seeks a structural pattern, or more precisely, the lack of equivalence between classes. Then for each pair of classes not defined as equivalent, it is checked whether the concepts they represent could be synonyms according to WordNet (Fellbaum, 1998), so that possible equivalences between classed are proposed to the user. For "P31. Defining wrong equivalent classes" exactly the opposite is checked, that is, whether two concepts that are defined as equivalent are not considered synonyms in WordNet (Fellbaum, 1998) in any context.

It should be noted that for some pitfalls, the detection methods applied might not cover all the possible situations in which a pitfall occurs but a subset of them. In these cases, the methods may be indicators, but for detecting non-simple

Figure 6. Example of patterns to detect pitfalls (part 1 of 2)

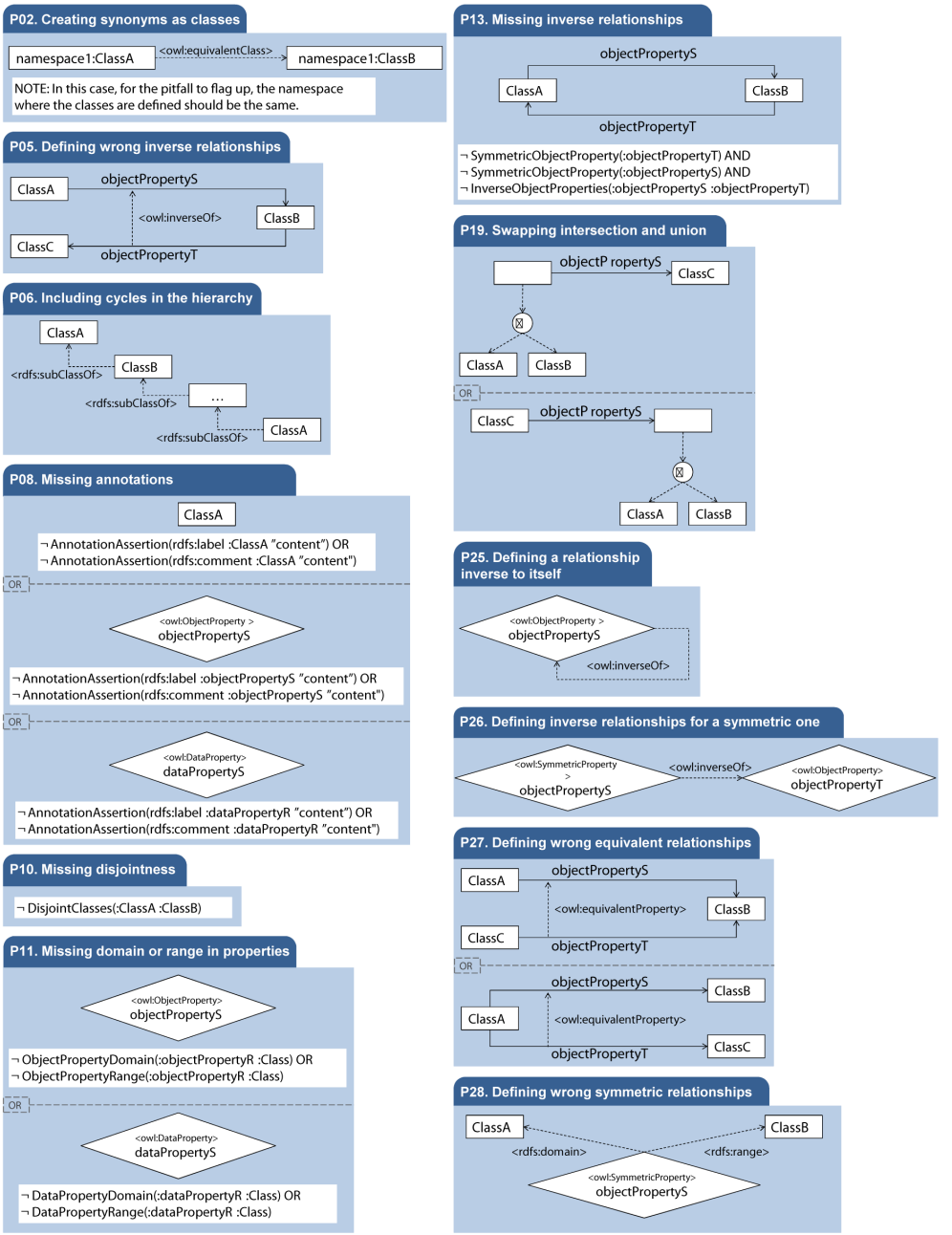


Figure 7. Example of patterns to detect pitfalls (part 2 of 2)

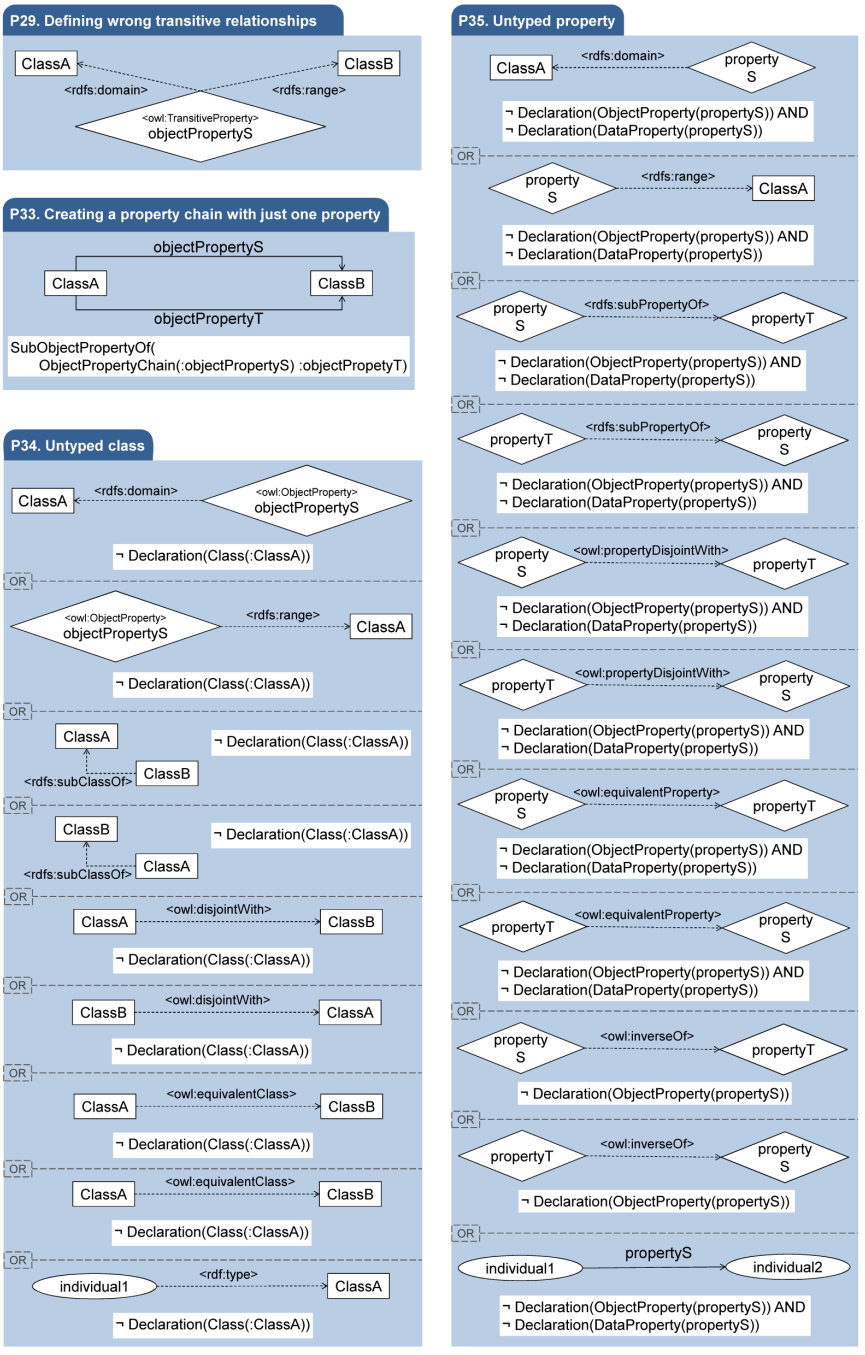
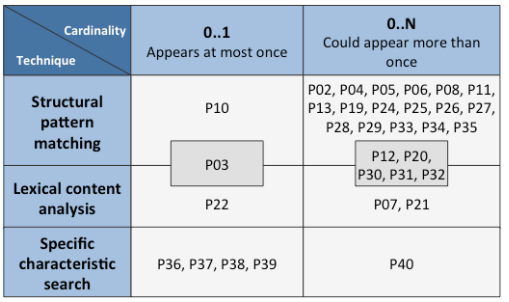


Figure 8. Classification of pitfalls based on the techniques used for their diagnoses



pitfalls background knowledge might be needed. For example, while "P11. Missing domain or range in properties" is detected in all possible cases by the pattern presented in Figure 6, it is not the case for "P05. Defining wrong inverse relationships". In this case, the current pattern will not cover the case of defining, in a math ontology, the relationship "lessThan" as inverse of "greaterThan" instead of "lessThanOrEqual", as some background and common sense knowledge is needed. We plan to improve these methods by incorporating linguistic techniques and resources as proposed in Suárez-Figueroa, Kamel, and Poveda-Villalón (2013).

In other cases, a detected pitfall might not represent a factual error, and this might be due to specific modeling decision or requirements. For example, "P02. Creating synonyms as classes" might be implemented in some cases in order to support backwards compatibility between different versions of the same ontology.

MOST COMMON PITFALLS

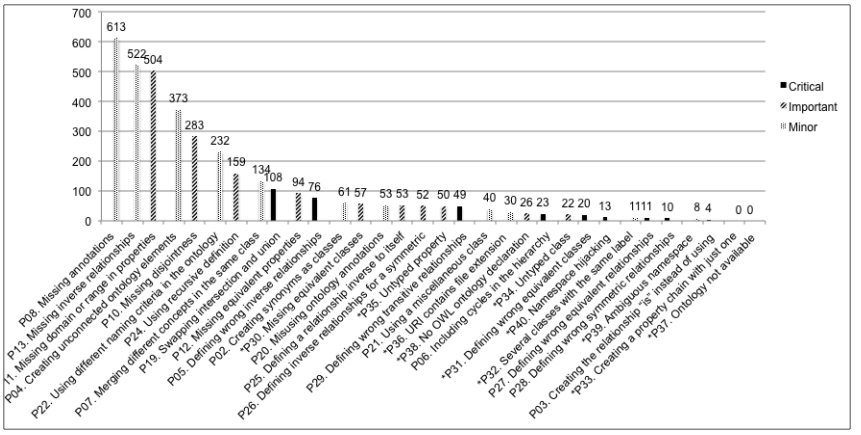
In order to know which are the most frequent errors in ontology development, we have recorded the number of pitfalls detected in each ontology analyzed with OOPS! To carry out this task we used the 32 pitfalls implemented up to February 2014.

When analyzing OOPS! execution logs, we could observe that

- Between November 14th, 2011 and February 17th, 2014, 1971 executions were carried out. During these executions, the ontology analyzed was identified by its URI in 1809 cases, whereas the ontology was "anonymous" (its URI was not defined or it was "null") in 162 cases.
- From these 1809 ontologies identified, some URIs indicate that the same ontology has been evaluated several times. We have filtered duplicated URIs, keeping only the first execution per URI. As a result, we counted 610 different ontologies. Further studies will take into account all the executions per URI and analyze the evolution of the pitfalls appearing.
- With regard to the 162 anonymous ontologies, we have removed executions with equal results, assuming that they belong to the same ontology, thus avoiding duplications. As a result, we counted 83 different anonymous ontologies.
- Overall, OOPS! has analyzed 693 ontologies[23] (610 with URI and 83 anonymous). This set of random ontologies submitted by OOPS! users contains upper level ontologies, as well as domain ontologies. These ontologies were developed either by domain experts, students, newcomers or ontology experts.

Finally, Figure 9 shows in how many ontologies each pitfall implemented in OOPS! has been diagnosed. The table reveals that most

Figure 9. Most frequent pitfalls diagnosed by OOPS! in a set of 693 ontologies



common pitfalls in ontologies are those related to the lack of explicit human and machine-readable information. However, these pitfalls do not correspond to those defined as critical by ontology practitioners but to those defined as "important" or "minor".

It should be noted that up to September 2013 only 21 were implemented and since then 11 new pitfalls have been implemented and included in the system, more precisely from P30 to P40, marked with a * in Figure 9. Therefore these new 11 pitfalls have been observed within 241 different ontologies instead of the 693 previously mentioned.

This study is complemented with a deeper analysis, described in Keet, Suárez-Figueroa, and Poveda-Villalón (2013), about pitfalls detected in (1) ontologies registered in OOPS! log; (2) ontologies developed by students; and (3) well-known ontologies developed by experts. In this analysis, the authors conclude that in most of the cases there is no clear evidence of noteworthy differences between the ontologies extracted from OOPS! log, the ones developed by students and the well-known ontologies. Therefore, even though the lack or appearance of pitfalls is considered a sign of quality, it could not be considered a measure of maturity in ontologies.

USER-BASED EVALUATION

OOPS! main goal is to get ontology evaluation closer to ontology developers, mainly newcomers and domain experts who are not familiar with description logics and ontology implementation languages.

In order to have an impression of the users' satisfaction when using OOPS!, a feedback form[24] is available online. On this form users can express their impressions after using the system. The answers to the questionnaire received so far reveal that (a) the tool clearly shows which is the problem detected; (b) OOPS! is a useful system; and (c) users would use it again and recommend it to their colleagues. Some users also pointed out some drawbacks such as (a) only *rdfs:label* and *rdfs:comment* are considered as annotation but not *skos*[25] or *dc*[26] annotations; and (b) OOPS! does not provide suggestions about how to solve a problem.

In that questionnaire, users also indicated how the system effectively improved the ontologies and helped in the process of ontology curation. In this regard, users mainly pointed out that OOPS! was useful for (a) discovering potential missing statements (e.g. human readable annotations, domain and range declarations and property characterization as inverse, among others), (b) detecting incorrect pairs of inverse properties, (c) enriching property definitions

(e.g. by adding the symmetric or transitive characteristic). Besides being used to diagnose ontologies, OOPS! has also been useful as part of the ontology assessment process in the context of ontologies for human behavior recognition, as explained in Rodríguez et al. (2014).

We have also received feedback and suggestions by email in which users show their agreement or disagreement regarding, for example, (a) the pitfall "P13. Missing inverse relationships", which is one of the typical debatable modeling decisions; or (b) when any pitfall detected affects ontology elements that belong to an imported ontology. More detailed information about this type of user evaluation can be found at (Poveda-Villalón, Suárez-Figueroa, & Gómez-Pérez, 2012).

Next, we present some evidence of how OOPS! has been used and adopted worldwide[27] up to February 17th, 2014. To do so, we have analyzed the log files from the server (from March 1st, 2012 to February 17th, 2014). From these logs we have deduced that OOPS! homepage has been visited over 3000 times from 69 different countries, and that the system has been executed around 2000 times[28] from 48 countries. It should be noted that the total number of different IP addresses for accessing and executing OOPS! is 1446 and 535, respectively.

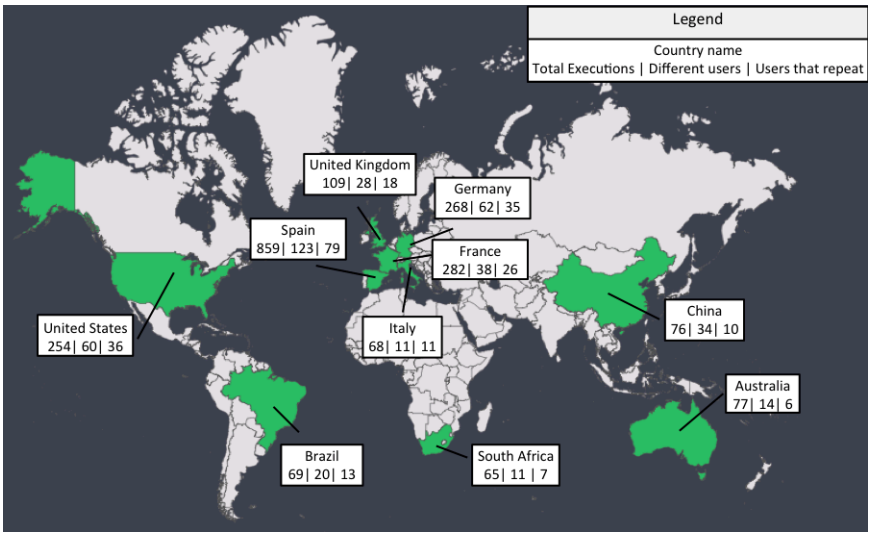
Focusing on the ten countries from where OOPS! has been executed most, Figure 10 shows how many times OOPS! has been executed in each country, how many single users have run it (different IPs), and how many users have executed the system more than once. These figures show that, in general, most of OOPS! users execute the system more than once.

It is worth mentioning that the OOPS! web service[29] has been integrated by third-party software; more precisely, it has been integrated into the Ontohub repository[30]. Finally, the system has been distributed for local installation within some private enterprises since their security policies do not allow them to submit the ontologies to an external website.

RELATED WORK

While in the introduction of this work a number of methods and techniques on ontology evaluation have been reviewed, in this section we focus on existing tools. More precisely, we review topology-based tools for ontology evaluation, that is, those tools focused on the internal

Figure 10. Map with the top 10 countries executing OOPS!



structure (classes, properties, instances, and the explicit and formal relations between them) of the ontology. Basic systems as syntax validators (e.g., RDF Validation Service[31] or Manchester OWL Validator[32]) are out of scope as they only check whether the ontology is compliant with the given ontology implementation language.

There are systems that depend on an associated ontology editor. This is the case of XD-Analyzer[33], a plug-in for NeOn Toolkit[34], and Ontocheck[35] (Schober et al., 2012), a plug-in for Protégé. The former checks some structural features (such as lack of domain and range definitions, use of intersection in domain and ranges, isolated entities, lack of annotations, and missing types, among others) and architectural features (e.g., unused imported ontologies), whereas the latter focuses on metadata aspects (e.g., annotations and naming conventions). The wiki-based ontology editor Moki[36] (Pammer, 2010) also provides some evaluation features (e.g., lack of annotation and orphaned elements). In addition, even though there is some overlap, the number of problems detected by these tools is lower than the current list of pitfalls detected by OOPS!. In order to provide a detailed comparison, Table 3 shows which pitfall could be detected by OOPS! and the different tools mentioned above. In addition, the right column shows which pitfall could be detected by means of a reasoner and specific test cases designed to identify each type of error.

Regarding web-based systems, we can consider OQuaRE[37], which extracts quality measurements from the ontology structure and compares these measurements to certain predefined values. The main drawbacks of this tool are that it does not point out any specific problem and that it does not give any information about how to improve the ontology.

Finally, we can mention command-line tools such a Eyeball[38], which is also available as Java API, a fact that makes its use more suitable for users with technological background. A graphical user interface in the form of a desktop application is also provided; however, the interface is still in an experimental phase. On the other hand, the problems detected by this tool have little overlap with OOPS!. Its main drawbacks are the technical knowledge needed to use it and the installation process required.

CONCLUSION AND FUTURE WORK

Evaluating an ontology that is being designed is a vital activity in any ontology development project. A number of approaches for ontology evaluation and tools have been proposed in the literature in the last decades.

In this work we have focused on a diagnosis method based on a checklist of common errors against which the ontology is compared. Our first contribution, in the form of a live catalogue of pitfalls, represents an extension of previous works about common problems in ontologies.

The automation of the detection process of 32 pitfalls included in the catalogue leads us to our second contribution: OOPS! (OntOlogy Pitfall Scanner!), an online tool for (semi-) automatic ontology diagnosis. This tool aims to help developers, mainly newcomers, during the ontology evaluation activity. OOPS! represents a step forward within ontology evaluation tools since (a) it enlarges the list of errors detected by most recent and available systems, such as MoKi (Pammer, 2010), XD-Analyzer and OntoCheck (Schober et al., 2012); (b) it is fully independent of any ontology development environment; (c) it works with the main web browsers (Firefox, Chrome, Safari, and Internet Explorer); and (d) its modular design facilitates the inclusion or removal of detection methods. In addition, the system could also be used for ontology selection. For example, when an organization wants to publish an existent data set as LD, the publisher has to choose one or more ontologies to model the published data. In this case, OOPS! could be used to compare the candidate ontologies along different quality dimensions.

It can be stated that the approach here presented has been widely accepted by the semantic web community and experts in other areas. Our approach is supported by the following facts:

*Table 3. Comparative of pitfall coverage between tools (✓ pitfall covered by the tool - * pitfall that could be detected by a reasoner and specific test cases)*

Pitfall	OOPS!	XD-Tools	Moki	Onto-check	Reasoner
P01. Creating polysemous elements					
P02. Creating synonyms as classes	✓				
P03. Creating the relationship "is" instead of using ''rdfs:subClassOf'', ''rdf:type'' or ''owl:sameAs''	✓				
P04. Creating unconnected ontology elements	✓	✓	✓		
P05. Defining wrong inverse relationships	✓				*
P06. Including cycles in the hierarchy	✓				*
P07. Merging different concepts in the same class	✓				
P08. Missing annotations	✓	✓	✓	✓	
P09. Missing basic information					
P10. Missing disjointness	✓				
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"					*
P16. Misusing primitive and defined classes					*
P17. Specializing a hierarchy exceedingly					
P18. Specifying the domain or range exceedingly					
P19. Swapping intersection and union	✓	✓			*
P20. Misusing ontology annotations	✓				
P21. Using a miscellaneous class	✓				
P22. Using different naming criteria in the ontology	✓			✓	
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	✓				*
P30. Missing equivalent classes	✓				*
P31. Defining wrong equivalent classes	✓				*
P32. Several classes with the same label	✓				
P33. Creating a property chain with just one property	✓				*
P34. Untyped class	✓				
P35. Untyped property	✓				

continued on following page

Table 3. Continued

Pitfall	OOPS!	XD-Tools	Moki	Onto-check	Reasoner
P36. URI contains file extension	✓				
P37. Ontology not available	✓				
P38. No OWL ontology declaration	✓				
P39. Ambiguous namespace	✓				
P40. Namespace hijacking	✓				

- OOPS! has been broadly accepted by a high number of users worldwide and has been executed more than 2000 times from 48 different countries.
- It has been continuously used from very different geographical locations.
- It is integrated with third-party software and locally installed in private enterprises (e.g., Semantic Arts[39] and Raytheon[40]).

To sum up, it could be stated that the approach proposed in this work has proof of being on the right track since it has become useful for ontology practitioners and for newcomers willing to evaluate their ontologies. All along the paper we have tried to show how both the catalogue and the tool are maintained and evolved according to users' feedback and research results.

Even though there are still several complex issues to address, our immediate future work will concentrate on the automation of the remaining 8 pitfalls and the enhancement of some of the already implemented ones. This extension might require increasing the users' interaction with the system by keeping them on the loop and using natural language processing techniques as proposed in Suárez-Figueroa, Kamel, and Poveda-Villalón (2013). Future lines of work should create and incorporate guidelines into OOPS! in order to repair the ontology according to the detected pitfalls.

Focusing on the LOD scenario in which a huge amount of data is annotated by making use of ontologies, an immediate line of work is to consider such data during the evaluation with the purpose of enhancing the results. As a first step, mismatches between the model defined and the instantiated data could be detected as well as inconsistencies.

Another line of work would involve making the system scalable for ontologies that contain a high number of terms. At the moment of writing this paper the system presents important delays with big ontologies, as for example DBpedia ontology[41], being the main bottleneck the number of object properties defined in the ontologies.

More ambitious plans include allowing users to define their own pitfalls or to contextualize existing ones and providing the mechanisms to interpret and process the pitfalls without manual encoding.

Finally, the integration of OOPS! within existing ontology editors, such as WebProtege[42] or the NeOn Toolkit, would be very convenient for the users since they would not need to change platforms to repair their ontologies after the diagnosis phase.

ACKNOWLEDGMENT

We are very grateful to Rosario Plaza for her revisions of the English language; to Antonio Jiménez and Alfonso Mateos for their comments and help on statistics; to Mariano Fernández-López for his revision and suggestions on the ontology part; to Miguel García for his technical support; to all the anonymous reviewers, and finally, to OOPS! users. This work has been partially supported by the Spanish project "4V: volumen, velocidad, variedad y validez en la gestión innovadora de datos" (TIN2013-46238-C4-2-R).

REFERENCES

Aguado de Cea, G., Gómez-Pérez, A., Montiel-Ponsoda, E., & Suárez-Figueroa, M. C. (2008). Natural language-based approach for helping in the reuse of ontology design patterns. In Knowledge engineering: Practice and patterns (pp. 32-47). Springer Berlin Heidelberg. doi:10.1007/978-3-540-87696-0_6

Allemang, D., & Hendler, J. (2011). *Semantic web for the working ontologist: Effective modeling in RDFS and OWL*. Elsevier.

Archer, P., Goedertier, S., & Loutas, N. (2012). *D7. 1.3–Study on persistent URIs, with identification of best practices and recommendations on the topic for the MSs and the EC*. PwC EU Services.

Barron, F. H., & Barrett, B. E. (1996). Decision quality using ranked attribute weights. *Management Science*, *42*(11), 1515–1523. doi:10.1287/mnsc.42.11.1515

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American*, *284*(5), 28–37. doi:10.1038/scientificamerican0501-34 PMID:11341160

Blomqvist, E., Gangemi, A., & Presutti, V. (2009). *Experiments on pattern-based ontology design*. In *Proceedings of the Fifth International Conference on Knowledge Capture* (pp. 41-48). ACM. doi:10.1145/1597735.1597743

Burton-Jones, A., Storey, V. C., Sugumaran, V., & Ahluwalia, P. (2005). A semiotic metrics suite for assessing the quality of ontologies. *Data & Knowledge Engineering*, *55*(1), 84–102. doi:10.1016/j.datak.2004.11.010

Djedidi, R., & Aufaure, M. A. (2010). ONTO-EVOAL an ontology evolution approach guided by pattern modeling and quality evaluation. In Foundations of information and knowledge systems (pp. 286-305). Springer Berlin Heidelberg.

Duque-Ramos, A., Fernández-Breis, J. T., Stevens, R., & Aussenac-Gilles, N. (2011). OQuaRE: A SQuaRE-based approach for evaluating the quality of ontologies. *Journal of Research and Practice in Information Technology*, *43*(2), 159.

Fellbaum, C. (1998). WordNet: An electronic lexical database. *WordNet*. Available at http://www. cogsci. princeton. edu/wn

Fernández-López, M., Gómez-Pérez, A., Sierra, J. P. & Sierra, A. P. (1999). Building a chemical ontology using methontology and the ontology design environment. *Intelligent Systems and their Applications, IEEE, 14*(1), 37-46.

Gangemi, A., Catenacci, C., Ciaramita, M., & Lehmann, J. (2006). *Modelling ontology evaluation and validation* (pp. 140–154). Springer Berlin Heidelberg.

Gómez-Pérez, A. (2004). Ontology evaluation. In S. Staab, & R. Studer (Eds.), *Handbook on ontologies* (pp. 251–273). Springer. doi:10.1007/978-3-540-24750-0_13

Guarino, N., & Welty, C. A. (2009). An overview of OntoClean. In *Handbook on ontologies* (pp. 201–220). Springer Berlin Heidelberg. doi:10.1007/978-3-540-92673-3_9

Heath, T., & Bizer, C. (2011). Linked data: Evolving the web into a global data space. *Synthesis Lectures on the Semantic Web: Theory and Technology, 1*(1), 1-136.

Hogan, A., Harth, A., Passant, A., Decker, S., & Polleres, A. (2010). *Weaving the pedantic web*. In Linked Data on the Web Workshop.

Ji, Q., Haase, P., Qi, G., Hitzler, P., & Stadtmüller, S. (2009). RaDON—repair and diagnosis in ontology networks. In *The semantic web: Research and applications* (pp. 863–867). Springer Berlin Heidelberg.

Keet, C. M., Suárez-Figueroa, M. C., & Poveda-Villalón, M. (2013). The current landscape of pitfalls in ontologies. In *Proceedings of the 5th International Conference on Knowledge Engineering and Ontology Development*.

Miettinen, K. (1999). *Non-linear multiobjective optimization*. Kluwer Academic Publishers.

Noy, N. F., & McGuinness, D. L. (2001*). Ontology development 101: A guide to creating your first ontology*. Technical Report SMI-2001-0880, Standford Medical Informatics.

Pammer, V. (2010) *PhD thesis: Automatic support for ontology evaluation review of entailed statements and assertional effects for OWL ontologies*. Engineering Sciences. Graz University of Technology.

Pinto, H. S., Staab, S., & Tempich, C. (2004). DILIGENT: Towards a fine-grained methodology for distributed, loosely-controlled and evolvInG. In *Proceedings of the 16th European Conference on Artificial Intelligence (Ecai 2004) (Vol. 110*, p. 393). IOS Press.

Poveda-Villalón, M., Suárez-Figueroa, M. C., & Gómez-Pérez, A. (2010). A double classification of common pitfalls in ontologies. In *Workshop on Ontology Quality at the 17th International Conference on Knowledge Engineering and Knowledge Management*.

Poveda-Villalón, M., Suárez-Figueroa, M. C., & Gómez-Pérez, A. (2012). Validating ontologies with oops! In *Knowledge Engineering and Knowledge Management* (pp. 267–281). Springer Berlin Heidelberg. doi:10.1007/978-3-642-33876-2_24

Poveda-Villalón, M., Vatant, B., Suárez-Figueroa, M. C., & Gómez-Pérez, A. (2013). Detecting good practices and pitfalls when publishing vocabularies on the web. In *Proceedings of the Workshop on Ontology Patterns at the 12th International Semantic Web Conference*.

Presutti, V., Gangemi, A., David S., Aguado, G., Suárez-Figueroa, M.C., Montiel-Ponsoda, E. & Poveda, M. (2008) *NeOn D2.5.1: A library of ontology design patterns: Reusable solutions for collaborative design of networked ontologies*. NeOn project. (FP6-27595).

Rector, A., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., & Stevens, R. et al. (2004). OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In *Engineering Knowledge in the Age of the Semantic Web* (pp. 63–81). Springer Berlin Heidelberg. doi:10.1007/978-3-540-30202-5_5

Rodríguez, N. D., Cuéllar, M. P., Lilius, J., & Calvo-Flores, M. D. (2014). A survey on ontologies for human behavior recognition. *ACM Computing Surveys*, *46*(4), 43. doi:10.1145/2523819

Schober, D., Tudose, I., Svatek, V. & Boeker, M. (2012). OntoCheck: Verifying ontology naming conventions and metadata completeness in Protégé 4. *Journal of Biomedical Semantics, 3*(Suppl 2), S4.

Staab, S., Studer, R., Schnurr, H. P., & Sure, Y. (2001). Knowledge processes and ontologies. *IEEE Intelligent Systems*, *16*(1), 26–34. doi:10.1109/5254.912382

Strasunskas, D., & Tomassen, S. L. (2008). The role of ontology in enhancing semantic searches: The EvOQS framework and its initial validation. *International Journal of Knowledge and Learning, 4*(4), 398-414.

Suárez-Figueroa, M. C. (2010) *PhD Thesis: NeOn methodology for building ontology networks: Specification, scheduling and reuse.* Universidad Politécnica de Madrid.

Suárez-Figueroa, M. C., Cea, G. A. D., & Gómez-Pérez, A. (2013). Lights and shadows in creating a glossary about ontology engineering. *Terminology*, *19*(2), 202–236. doi:10.1075/term.19.2.03sua

Suárez-Figueroa, M. C., Gómez-Pérez, A., Motta, E., & Gangemi, A. (Eds.). (2012). *Ontology engineering in a networked world*. Springer. doi:10.1007/978-3-642-24794-1

Suárez-Figueroa, M. C., Kamel, M., & Poveda-Villalón, M. (2013). Benefits of natural language techniques in ontology evaluation: The OOPS! case. In *Proceedings of the 10th International Conference on Terminology and Artificial Intelligence (TIA 2013)* (pp. 107-110). ISBN: 978-2-9174-9025-9.

Winkler, R. L. & Hays, W. L. (1985). *Statistics: Probability, inference, and decision*.

ENDNOTES

1 http://www.w3.org/DesignIssues/LinkedData.html

2 http://neon-toolkit.org/wiki/XDTools

3 http://protegewiki.stanford.edu/wiki/OntoCheck

4 https://moki.fbk.eu/website/index.php

5 It should be observed that the term "pitfall" is used all along this paper for characteristics that often represent a problem or that could lead to errors in ontologies; however, this is not always the case. In other words, depending on the ontology at hand, pitfalls can or cannot represent an actual error.

6 http://www.w3.org/TR/owl-ref/

7 http://www.w3.org/TR/rdf-primer/

8 The online version of the catalogue is available at http://www.oeg-upm.net/oops/catalogue.jsp. Previous versions were included in Poveda-Villalón, Suárez-Figueroa, and Gómez-Pérez (2010) and Poveda-Villalón, Suárez-Figueroa, and Gómez-Pérez (2012).

9 Pitfalls "17. Specializing a hierarchy exceedingly" and "P18. Specifying the domain or range exceedingly" were previously titled "P17. Specializing too much a hierarchy" and "P18. Specifying too much the domain or the range" respectively.

10 The call was launched through several mailing list used by the semantic web community and through particular emails sent to known OOPS! users, mainly experts on ontology modeling or evaluation.

11 It is worth mentioning, since it could seem contradictory, that for processing the data and ranking the pitfalls we have assigned the value 3 for critical pitfalls, so that they appear in the top positions. However, for assigning importance levels within the catalogue we have set the "critical" position in 1, since the

critical pitfalls should be corrected in first place.

12 See file "SurveyImportanceLevelsLexicographicOrder.pdf" at http://goo.gl/0IkbS2.

13 The data and calculations for obtaining the coefficients are available at http://goo.gl/QeSyHX

14 Figure 3 also indicates which pitfalls are currently implemented by OOPS!.

15 http://www.oracle.com/technetwork/java/javaee/overview/index.html

16 http://www.w3.org/html/wg/

17 http://jquery.com/

18 http://www.oracle.com/technetwork/java/javaee/jsp/index.html

19 http://www.w3.org/Style/CSS/

20 http://jena.sourceforge.net/

21 At the moment of writing this document no inference is used during the evaluation process.

22 http://oops-ws.oeg-upm.net/

23 The filtered data, that is, without duplicates, from OOPS! log is available at http://goo.gl/DWSTNW. Due to privacy issues the ontologies' URIs have been renamed.

24 http://goo.gl/9W7bLl

25 skos is the prefix used for the namespace http://purl.org/linked-data/xkos#

26 dc is the prefix used for the namespace http://purl.org/dc/terms/

27 Detailed use statistics are available at http://www.oeg-upm.net/oops/use.html.

28 It is worth mentioning that these executions are those registered in the server log since May 2012. This log is different from the OOPS! log of executions that gathers ontologies and results since November 2011.

29 http://oops-ws.oeg-upm.net/

30 See http://goo.gl/TKHr5z for more information.

31 http://www.w3.org/RDF/Validator/

32 http://owl.cs.manchester.ac.uk/validator/

33 http://neon-toolkit.org/wiki/XDTools

34 http://neon-toolkit.org/wiki/Main_Page

35 http://protegewiki.stanford.edu/wiki/OntoCheck

36 https://moki.fbk.eu/website/index.php

37 http://miuras.inf.um.es:9080/oqmodelsliteclient/

38 http://jena.sourceforge.net/Eyeball/

39 http://semanticarts.com/

40 http://www.raytheon.com/

41 http://wiki.dbpedia.org/Ontology

42 http://protegewiki.stanford.edu/wiki/WebProtege

María Poveda-Villalón is a Ph.D student at the Artificial Intelligence Department of the Computer Science Faculty of Universidad Politécnica de Madrid, in the Ontology Engineering Group. Her research activities focus on Ontological Engineering, Knowledge Representation and the Semantic Web. Previously she finished her studies as an engineer in Computer Science (2009) by Universidad Politécnica de Madrid, and then she moved to study the Artificial Intelligence Research Master finished in 2010 in the same university. She has collaborated during a four-month research stay in 2013 with Mondeca (París, France), during a three-month stay in 2012 with the Free University of Berlin and with the University of Liverpool in a three-month stay in 2011.

Asunción Gómez-Pérez is Full Professor at UPM (2007), Director of the Artificial Intelligence department (2008), Director of the Ontology Engineering Group (1995, 8th group in the UPM ranking), Director of the Master (2010) and Ph.D Program (2013) on Artificial Intelligence, Director of the Co-founder of the COM joint institute between Santander Bank and UPM (2012). PhD in Computer Science (1993) and Master on Business Administration (1992). Before joining UPM, she was visiting (1994-1995) as a postdoc the Knowledge Systems Laboratory at Stanford University. She has supervised 18 Ph.D thesis, she has coordinated 4 EU projects SEALS, SemSorGrid4Env and Ontogrid and now she is coordinating LIDER. She has participated 21 EU projects (FP5, FP6 and FP7) as main researchers, and in more than 40 national projects funded by Spanish research agencies and companies. Her main research interests are ontologies, semantic technologies, linked data and the semantic Web. She has published more than 150 papers and two books on Ontological Engineering. Her works on Ontological Engineering about Methontology and the NeON methodology are world-wide known. She has been co-director of the summer school on Ontological Engineering and the Semantic Web since 2003 up to 2011. She acts as reviewers in journals and conferences related with semantic technologies.

Mari Carmen Suárez-Figueroa is a teaching assistant at the Escuela Técnica Superior de Ingenieros Informáticos, Universidad Politécnica de Madrid (UPM) and a senior researcher at the Ontology Engineering Group. She graduated in Computer Science in 2001 and got the PhD in Artificial Intelligence in 2010. She has received an Outstanding Award granted by the UPM PhD Commission. Her research lines include ontology development methodologies, ontology development in different domains, ontology evaluation, ontology design patterns, and linked data. In these areas, she has participated in several European and Spanish projects. She has been research visitor at University of Liverpool in 2004, at KMi (Open University) in 2007, and at IRIT (Toulouse) in 2012. She is co-editor of the book "Ontology Engineering in a Networked World" (Springer 2012). She co-organized sessions, conferences, workshops, and tutorials in international events such as ESWC 2014, TKE 2012, ISWC 2012, EKAW 2012, EKAW 2008, ESWC 2008, and WWW 2006.