ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
Επιστήμη του Διαδικτύου
«Web Science»

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

"Ανάπτυξη εργαλείου αξιολόγησης οντολογιών με χρήση ποσοτικών δεικτών, ως πρόσθετου του περιβάλλοντος Protégé"

Γεώργιος Τάντσης

ΕΠΙΒΛΕΠΩΝ: Διονύσιος Κεχαγιάς
Μεταδιδακτορικός ερευνητής στο Ινστιτούτο Πληροφορικής και Τηλεματικής (Ι.Π.ΤΗΛ.) του Εθνικού Κέντρου Έρευνας και Τεχνολογικής Ανάπτυξης (Ε.Κ.Ε.Τ.Α.)

Θεσσαλονίκη, Ιούλιος 2013

ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
Επιστήμη του Διαδικτύου
«Web Science»

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

"Ανάπτυξη εργαλείου αξιολόγησης οντολογιών με χρήση ποσοτικών δεικτών, ως πρόσθετου του περιβάλλοντος Protégé"

Γεώργιος Τάντσης

ΕΠΙΒΛΕΠΩΝ: Διονύσιος Κεχαγιάς
Μεταδιδακτορικός ερευνητής στο Ινστιτούτο Πληροφορικής και Τηλεματικής (Ι.Π.ΤΗΛ.) του Εθνικού Κέντρου Έρευνας και Τεχνολογικής Ανάπτυξης (Ε.Κ.Ε.Τ.Α.)

Εγκρίθηκε από την Τριμελή Εξεταστική Επιτροπή την 15η Ιουλίου 2013.

……	……	……
Ιωάννης Αντωνίου Καθηγητής Α.Π.Θ.	Πολυχρόνης Μωυσιάδης Καθηγητής Α.Π.Θ.	Παν. Μπαμίδης Επ. Καθηγητής Α.Π.Θ.

Θεσσαλονίκη, Ιούλιος 2013

……..

Γεώργιος Τάντσης

Πτυχιούχος Διοίκησης Τεχνολογίας Πανεπιστήμιο Μακεδονίας

Περίληψη

Το παρόν έγγραφο αποτελεί μεταπτυχιακή διπλωματική εργασία για τη δημιουργία ενός πρόσθετου (plug-in) για το πρόγραμμα συγγραφής οντολογιών Protégé με σκοπό την αξιολόγηση οντολογιών. Αρχικά παρουσιάζονται οι σημαντικότερες μεθοδολογίες αξιολόγησης οντολογιών που υπάρχουν στη βιβλιογραφία και γίνεται μια συνοπτική αξιολόγηση αυτών. Στη συνέχεια, ακολουθεί η ανάπτυξη μιας νέας μεθοδολογίας αξιολόγησης οντολογιών, η οποία περιλαμβάνει μια σειρά από δείκτες αξιολόγησης και τις αντίστοιχες μετρικές αυτών. Το κίνητρο πίσω από την δημιουργία του προτεινόμενου πλαισίου αξιολόγησης είναι να παρέχει τα μέσα για τη βελτίωση των υπαρχουσών οντολογιών και την ανάπτυξη νέων που να τηρούν ένα σύνολο βέλτιστων πρακτικών, συμπεριλαμβανομένων της αρτιότερης δομής, την αύξηση της αναγνωσιμότητας και του περιορισμού των πλεονασματικών στοιχείων αυτών. Με βάση την προαναφερθείσα προτεινόμενη μεθοδολογία αξιολόγησης οντολογιών αναπτύσσεται ένα πρόσθετο για το Protégé (software). Επίσης, γίνεται παρουσίαση του πρόσθετου (plug-in) αξιολόγησης οντολογιών εξηγώντας την λειτουργία των συστατικών μερών και της διεπαφής αυτού. Επιπλέον, μέσα από μια απλή μελέτη περίπτωσης παρουσιάζεται η χρήση του plug-in σε πραγματικές οντολογίες δείχνοντας την προστιθέμενη αξία αυτού. Τέλος, υπάρχει ένας βήμα προς βήμα οδηγός που επεξηγεί την διαδικασία που λαμβάνει χώρα προκειμένου να δημιουργηθεί ένα plug-in για το Protégé.

Λέξεις Κλειδιά

Αξιολόγηση Οντολογιών, Πρόσθετο (plug-in), Protégé, Οδηγός

Abstract

This document is a master's thesis for the creation of the “Ontology Evaluation plug-in for the Protégé (software)”. At the beginning the state of the art on ontology evaluation methodologies that are found in the literature are described. Then it is introduced a new ontology evaluation methodology which comprises a set of evaluation criteria and the corresponding metrics. The motivation behind the development of the proposed evaluation framework is to provide the means for the improvement of existing ontologies and the development of new ones that adhere to a set of best practices, including efficient structure, increased readability and limited redundancy. Upon the aforementioned methodology the creation of the “Ontology Evaluation plug-in for the Protégé (software)” is based. Also, the “Ontology Evaluation plug-in for the Protégé (software)” is presented by explaining its interface and the usage of its components. Furthermore the plug-in is tested on a set of various ontologies, through a simple case study scenario, that shows the added value of the plug-in by showing how the proposed evaluation framework can be applied for improving existing ontologies or facilitating the design process. Finally, there is also a tutorial that gives a step by step explanation of the procedure that takes place in order to create a Protégé plug-in.

Key Words

Ontology Evaluation, Protégé, plug-in, tutorial

Table of Contents

1 - Introduction

In computer science and information science, an ontology formally represents knowledge as a set of concepts within a domain, and the relationships between pairs of concepts. It can be used to model a domain and support reasoning about entities. In theory, an ontology is a "formal, explicit specification of a shared conceptualization". [1] Ontologies are the structural frameworks for organizing information and are used in artificial intelligence, the Semantic Web, systems engineering, software engineering, biomedical informatics, library science, enterprise bookmarking, and information architecture as a form of knowledge representation about the world or some part of it.

Building an ontology or an ontology network from scratch is not always an easy process. Even though many visualization support tools are available that facilitate the various steps of the ontology lifecycle, the core development of an ontology remains a manual task that requires good knowledge of the domain to be modeled, as well as good modeling skills and experience. It is a common practice for knowledge engineers to work together with domain experts in order to build robust ontologies. One very important decision that has to be made is whether the ontology will be built from scratch, or will reuse other resources (ontological or non-ontological). Reusing existing ontologies may save significant effort and facilitates interaction with different development tools. Since ontology authoring and design styles vary considerably, and generally speaking ontology authoring is an open and flexible process, ontology refinement is often necessary in order to improve badly formed ontologies.

In order to refine and change an ontology you've got first to evaluate it. This paper introduces a formal ontology evaluation framework that comprises a set of evaluation criteria and corresponding evaluation metrics in measurable form. The motivation behind the development of the proposed evaluation framework is mainly to overcome the lack of formal evaluation procedures along with concrete implementation tools, also providing efficient means for the implementation of ontology design best practices adhering to state of the art approaches and tools. The newly introduced evaluation framework aims at facilitating the ontology design process, providing the ontology designer with the means for evaluating existing ontologies or new ones with respect to a set of best practices, such as appropriate structure, increased readability and limited redundancy.

The tool that we introduce and implements the aforementioned framework is a Protégé *plug-in.*

What is Protégé ?

Protégé is a free, open source ontology editor and a knowledge acquisition system [2]. Like Eclipse, Protégé is a framework for which various other projects suggest plugins. This application is written

in Java and heavily uses Swing to create the rather complex user interface. Protégé recently has over 200,000 registered users.
Protégé is being developed at Stanford University in collaboration with the University of Manchester and is made available under the Mozilla Public License 1.1.

Why was the ontology evaluation plug-in developed ?

Looking at various professions you can see that people use some tools that without them it would have been impossible to successfully carry their jobs. Take for example a doctor that has to examine a patient's situation with a possible broken limb. The first thing that the doctor does is to take an X-ray scan in order to evaluate the patient's situation. Or think about a microbiologist, without a microscope it would have been impossible for him to work.

Searching the Protégé plug-ins repository you can find several of them that cover different needs. But if you want to evaluate your ontology's situation and have a complete picture of the key elements which is consisted there seems to be a lack of such a plug-in. So when other professionals, such as the doctors or the microbiologists mentioned before, have the means to cover their needs the Protégé user seems to be lacking a very useful tool. That was the motive that pushed us to develop the ontology evaluation plug-in, that we present in this thesis.

Thesis contributions

The main contributions of this thesis can be summarized as follows:

- It establishes a theoretical framework for evaluating ontologies.
- It introduces a set of metrics that allow for quantitative evaluation of ontologies.
- It proposes a set of good practices for ontology development.
- It enables comparison of existing ontologies with respect to the quantitative metrics in terms of their adherence to the aforementioned proposed practices.
- It develops a new protégé plug-in for adding ontology evaluation capabilities to the Protégé ontology authoring tool.
- It constructs a set of rules for implementing appropriate recommendations that are used for fine-tuning ontology that suffer from bad design practices.
- It showcases the operation of the developed plug-in by testing it to existing ontologies.

Structure of Thesis

The thesis is structured as follows:

Chapter 2 presents the various ontology evaluation methodologies that exist in the literature and a brief explanation of its one of them is given. However none of them perfectly fulfills all practical ontology evaluation needs.

Chapter 3 introduces a formal ontology evaluation methodology which comprises a set of evaluation criteria and the corresponding metrics. The methodology fills the gap found in the literature for a more integrated ontology evaluation.

Chapter 4 presents an implementation of a software tool in the form of a Protégé plug-in that executes the proposed evaluation process found in chapter 3. An explanation of the interface and the consisting parts of the plug-in is given.

Chapter 5 tests the plug-in on a set of various ontologies, through a simple case study scenario, that shows the added value of the plug-in by showing how the proposed evaluation framework can be applied for improving existing ontologies or facilitating the design process.

Chapter 6 is the conclusion that summarizes our study and presents thoughts for future work.

At the end of this document there is also a tutorial that gives a step by step explanation of the procedure that takes place in order to create a Protégé plug-in.

2 - State of the art on ontology evaluation methodologies

Various ontology evaluation methodologies exist in the literature; however none of them perfectly fulfills all practical ontology evaluation needs. Most of them are tailored to specific application requirements. In general the majority of the evaluation methodologies are classified into one of the following categories, according to the way in which the evaluation process is conducted [7]:

- Evaluation is based on how well an ontology fulfills the requirements of a particular application (e.g. work by Porzel-Malaka [8], Kalfoglou & Hu [37]).

- Evaluation is based on the comparison of ontologies to a "golden standard", which may itself be an ontology. This category includes OntoClean [9], OntoNews [39] as well as work by Dellschaft et al [10], Brank et al. [7] and Zavitsanos et al. [11].

- Evaluation involves comparisons of ontologies with a certain data source, such as a collection of documents about the domain to be covered. Examples include EvaLexon [12], OntoKhoj [38], work by Brewster et al. [13], Daelemans et al. [14] and Murdock et al. [15].

- Evaluation is done by human experts who try to assess how well the ontology meets a set of predefined set of attributes specified by certain criteria, standards, requirements, etc. This category includes the Ontometric [16], oQual [17], Pan-Onto-Eval [31], AKTiveRank [32], ODEval [33], OntoQA [35] and ONTOSTEVAL [18] evaluation frameworks, as well as the ones developed by Supekar [19], Burton-Jones et al. [20], Good and Tennis [21], Mostowfi and Fatouhi [34] and Tao et al. [22].

- All of the above methodologies are manual activities for evaluating ontologies. However, there are some efforts proposing semi-automatic techniques like Bioportal [23], Ontology Metrics [24], LExO [25], ROD [26], OCC [30], OntoManager [36] as well as the ones developed by Lehmann et al. [27] and Iannone et al. [28]. What follows is a description of the ontology evaluation methodologies based on the above classification.

2.1 - Application-based approaches

The first category relies on the fact that ontologies are typically consumed by various sorts of applications or tasks. When embedded in applications, ontologies might be a crucial factor significantly influencing the performance of the application on a given task. Thus, ontologies may be evaluated simply by plugging them into an application and evaluating how the ontology

performs within the application. Porzel and Malaka [8] adopted such an approach, a task-based evaluation process, in order to evaluate a set of ontologies. What can be seen as a drawback about this category is that the range of tasks needed to provide enough coverage for the typical applications, in which ontologies are used is too large, making methods of this category impractical to use in real cases. Additional identified drawbacks include: (a) the evaluation may be insensitive as in some cases the ontology could be only a small component of the application and therefore does not significantly influence the performance of the application; (b) in order for the evaluation framework to be trusted and reliable, comparison of ontologies must be accomplished under the same conditions, i.e., incorporating them into the same application, which is difficult to carry out due to different ontology formats and structures.

Kalfoglou & Hu [37] discuss the need to evaluate ontologies available from the SemanticWeb, from technological, strategic and political viewpoints. From their experience in building the Computer Science AKTive Space (Shadbolt et al., 2004), which is a portal to explore semantically-annotated domains, the authors emphasize problems with and suggest solutions for ontology evaluation. Problems include verifying appropriateness, validity, consistency, and so on. Solutions provided over the medium term are: the participation of experts in communities of practice (Lave & Wenger, 1991; Brown & Duguid, 2000) and ontology classification according to use. Similarly, solutions provided over the long term are related to the evolution of standardization and certification of ontology processes.

2.2 - “Golden standard”- based approaches

In the second category, Welty et al. [9] propose OntoClean for ontology evaluation. They suggest providing each concept and relation with various meta-properties, such as rigidity, identity, dependence and unity, which would help to discover inconsistency or ambiguity existing in ontologies. The main disadvantage of this approach is that a user needs to manually add all the meta-properties for concepts of the ontology, thus making the evaluation process rather tedious and difficult. On the contrary, our methodology omits the use of any type of meta-properties and this makes it easily applicable to any ontology. In addition, the number of actually tagged ontologies is obviously far too low. This again points out a discrepancy between the expected work and the expected benefit of using OntoClean. Another drawback of OntoClean is that tagging an ontology implies further ontological decisions possibly unintended by the ontology authors. Our approach dodges this problem by using objective, quantitative refinement criteria instead of an individual or small group’s subjective point of view.

Also in this category, work by Dellschaft and Staab [10] attempts to realize a comparison of two ontologies both at data and conceptual level. The first step of their work was to build a “golden standard” domain ontology, starting from the automatically learned concepts and relations. Then the automatically created ontology is assessed against the “golden standard” by using metrics such as

lexical and taxonomic precision and recall. A somewhat similar approach was discussed in [7]. The difference is that this one assumes that both ontologies are constructed over the same set of instances. Moreover, this approach completely disregards concept labels, but in many practical situations these labels are an important part of the ontology and contain adequate knowledge about the problem domain. Another approach [11] contributed also to the "golden-based" ontology evaluation methods, through a framework and a set of evaluation measures in order to assess learned ontologies against "golden standard" ontologies.

Maynard et al. [39] propose metrics for an ontology-based evaluation of information extraction. The comparison standard is composed of a text corpus, called OntoNews, enriched by annotations provided by a general purpose ontology. The authors describe traditional information extraction metrics: Precision and Recall (PR), Cost-based Evaluation (CE), Learning Accuracy (LA); and present a new proposal, in fact an LA extension, called Balanced Distance Metric (BDM). Three metrics are used to evaluate the standard: LA, DBM and a combination of PR and CE. The results indicate that both DBM and LA metrics perform better in extracting information from hierarchical structures, however problems are found when humans evaluate them.

After reviewing the aforementioned five methods the conclusion is that it is rather pointless to use "golden standard"-based methods during ontology evaluation. In practical tasks, where users have to choose the most suitable ontology in order to incorporate it into their system a "golden standard"-based method seems to be useless. Indeed, knowledge of the "golden standard" is not necessarily the ideal choice, since it is the outcome of manual, error prone process. Furthermore, any modeling problems associated to this "golden standard" will be reproduced through the evaluation method, which will reward ontologies with similar problems and penalize those which do not include related concepts. This also implies that the "golden standard"-based evaluation method lacks generalization as it always depends on the "golden standard" to be used.

2.3 - Approaches based on comparison with existing data sources

In the third evaluation category an ontology is assessed by comparison to existing data sources about the problem domain, to which the ontology refers. Such data source is usually a collection of textual documents. For example, EvaLexon main's goal is to evaluate at development time ontologies that are created by human experts from text [12]. In sharp contrast with OntoClean, EvaLexon is intended for linguistic rather than conceptual evaluation. Linguistic evaluation was based on precision and recall, comparing ontologies with statistically relevant terms. The obtained results seem to be subjective because the reference point for the evaluation is the text itself. If the ontology is tested against a different data source, the precision and recall that are used in order to evaluate the response of the ontology to a query engine will be different. Hence, this evaluation method cannot be generalized as it is based on a particular source of data to compare. However, the aim of a complete evaluation framework is to assess ontologies based on a common ground that

includes metrics relevant to the ontology structural and functional characteristics. This is the main difference that was identified between EvaLexon and the approach presented next to this document.

Another data-driven approach has been proposed in [13]. However, an evident drawback of this method is that it is difficult to guess about a type of relations between existing textual data and ontology terms, and, therefore, ontology evaluation seems to be rather complicated and obscure. In the case of more extensive and sophisticated ontologies that incorporate a lot of factual information, an approach in [14] points out how recently developed natural language processing (NLP) techniques can be used for evaluating ontologies' semantics (vs. their syntax). However, human intervention is still essential to evaluate the proposed ontological structures making this method rather slow and difficult.

OntoKhoj [38] is a portal featuring search, ranking, aggregation and classification services, focused on ontologies available on the Web. In order to determine the ontology subject and then classify it, textual data are extracted, such as the names of concepts and relations. Afterwards, these data are used as entries in a classification text model, trained with standard machine-learning algorithms.

Murdock et al. [15] reviewed evaluation methods that focus solely on syntactic correctness, on the preservation of semantic structure and on usability layer. They proposed two novel methods, the volatility and violation scores, for dynamic ontology evaluation and described the use of these methods for evaluating the different taxonomic representations that are generated at different times or with different amounts of expert feedback. The volatility score measures the structural stability over the ontology extension and evolution. The violation score measures the semantic fit between an ontology's taxonomic structure and the distribution of terms in an underlying text corpus. The main disadvantage of this approach is that it presents a complex mathematical analysis for these two scores. In addition, our methodology covers two more ontology layers, data/application and lexical/vocabulary layer.

2.4 - Attribute-based approaches

Last but not least family of ontology evaluation approaches, to which belongs the methodology that is recommended at the next chapter, deals with the problem of defining several metrics or attributes; for each metric, the ontology is evaluated and a numerical or linguistic score is assigned to it. In this family, work in [15] proposes that an ontology could be enhanced with metadata, such as its design policy, version number, the way it is being used by others, as well as peer reviews provided by ontology users. The downside of this approach is that it relies almost entirely on manual human intervention to both provide annotations and use them in order to evaluate and select an ontology.

The Ontometric method is based on a taxonomy of 160 metrics of ontologies [16], called multilevel framework of characteristics, that provides the outline to choose and to compare existing ontologies. The multilevel framework of characteristics has, in the superior level of the taxonomy,

five basic aspects on the ontologies that are denominated dimensions. These pertain to ontology content, its organization, the representation language and development tools among others. All these attributes must be filled-in by the user. The evident drawback of this system is a need of user manual assessment of ontologies, which is a complicated and time consuming task. Even for those attributes that have quantitative representation, such as Number_of_Axioms, Number_of_Concepts, Maximum_Depth, etc., Ontometric provides linguistic representations (i.e., non-numerical): such as low, medium, high, etc. This is an obvious drawback since it provides only an intuitive perception of the designated attributes, which is inadequate especially when it comes to perform concrete comparisons between two or more ontologies in terms of their numerical attributes, such as the number of concepts or the average depth.

A different approach [20] proposes ten simple evaluation criteria, such as syntactical correctness, clarity of vocabulary, lawfulness, richness, interpretability, consistency, accuracy, access history, authority, relevance. A drawback of this approach is that there is little documentation to help us ascertain to what extent the ontology matches the real-world state of the problem domain, to which it refers. Moreover, even though the criterion of accuracy against ontology queries is taken into account when computing the overall ontology score, it is usually difficult to compute the percentage of false statements in any other way apart from examining them all manually. On the positive side, we include the automatic calculation of the aforementioned criteria, as well as its support for metadata.

The method in [17] consists of a meta-ontology O2 that characterizes ontologies as semiotic objects. The meta-ontology is complemented with an ontology about ontology evaluation and validation, namely oQual. Based on O2 and oQual, this method identifies three main types of predefined criteria for ontology evaluation: structural criteria, which are typical of ontologies represented as graphs; functional criteria that are related to the intended use of an ontology and its components, i.e., their function; usability-related criteria, that depend on the level of annotation of the considered ontology. Although similar to the methodology that is presented on this paper, this method relies on the use of quite complex measures. This introduces a compounded complexity in this approach, which results in increasing the effort required for the ontology evaluation process.

On the other hand, the method proposed by Good and Tennis [21] define a collection of metrics for describing and comparing sets of terms in controlled and uncontrolled indexing languages. It is then shown how these metrics can be used to characterize a set of languages spanning folksonomies, ontologies and thesauri. This framework, although applicable to ontologies, is not designed for ontologies in principle, but for more generic forms of knowledge representation. One drawback of this approach is that it is difficult to construct automated tests to compare ontologies using the aforementioned criteria.

Tao et al. [22] have focused on instance ontology data evaluation. They have identified three categories of issues that may occur in instance data which are syntax errors, logical inconsistencies,

and several potential issues. Syntax errors are the issues indicating that the syntax representation of instance data does not conform to the corresponding syntax specifications such as RDF/XML, N3, N-Triple, OWL, etc; logical inconsistencies are the issues showing that the instance data includes contradictory axioms; finally, potential issues are the issues caused by the failure of the instance data to follow the restrictions that are imposed by several constraints on classes, properties and individuals. The downside of this approach is that it evaluates only instance data and not the whole knowledge base which includes classes, properties and instances.

Alani et al. [32] presents a technique called AKTiveRank that finds a set of related ontologies to a set of terms the user enters. It uses an aggregation of the values of the four measures AKTiveRank includes to evaluation ontology schemas to select one of the ontologies to be the most suitable. The measures they developed are: class match, density, semantic similarity, and betweenness. Corcho et al. [33] introduce the ODEval tool that can be used for the automatic detection of possible syntactical problems in ontologies, such as the existence of cycles in the inheritance tree of the ontology classes, inconsistency, incompleteness, and redundancy of classes and instances. Mostowfi and Fatouhi [34] define eight features they use to measure the quality of ontologies. These features are used to define a set of transformations to improve the quality of ontologies. For example, the authors suggest if a class (Student) has a property (Salary) that does not always have values (because it only holds for student assistants), then the class needs to be split into two: Student and Student Assistant. Other transformations attempt to make changes in properties or data types to make the ontology more consistent. OntoQA [35] works on populated ontologies, thus enabling it from utilizing knowledge represented in the instances to gain a better measure of the quality of the ontology. In OntoQA, metrics (features) are divided into two groups: schema metrics that address the design of the ontology schema and instance metrics that address the way instances are organized within the ontology. The first category evaluates ontology design and its potential for rich knowledge representation. The second category evaluates the placement of instance data within the ontology and the effective utilization of the knowledge modeled in the schema.

The Ontology Structural Evaluation Framework (ONTOSTEVAL) for benchmarking the internal graph structures was proposed in [14]. In this work, the simplicity of an adjacency matrix of a graph was used to compute the algebraic spectrum and structural dimensions of ontology. The framework was used in transport and biochemical ontologies. The corresponding adjacency, incidence matrices and other structural properties like depth, breadth and density were computed using MATLAB. What can be seen as a drawback in this approach is that it depends only on the structure of the ontology, disregarding other layers such as syntactic, application and usability.

The Pan-Onto-Eval [31] aims to extract a snapshot of an ontology that contains the most important characteristics of the ontology (concepts and relations that represent the thematic categories of the ontology). The measurements represent a comprehensive perspective on the following four issues: a) Triple Centricity: an ontology is meaningful when there are many diverse relationships, i.e., domain concepts associated with other concepts through diverse relations. Hence are analyzed their

roles with relations (i.e. whether they are domain or range concepts) and their importance. b) Theme Centricity: reflects the importance of non-ISA relations in the evaluation of any ontology in terms of relational richness. c) Structure Centricity: describes the topology (i.e., shape and size) of concept hierarchies of an ontology and d) Domain Centricity: an ontology may consist of more than one IS-A hierarchy that contribute to the semantics and distribution of information across the ontology.

After having reviewed the most representative ontology evaluation methodologies focusing on their comparison with our approach, we conclude that the vast majority of the examined approaches address different types of evaluation criteria. The only exceptions are Ontometric [16] and oQual [17] that share a set of common attributes that are handled as evaluation criteria. These two approaches, along with ours surpass the more traditional ontology evaluation approaches, such as "golden standard-" and data source based, in that they provide support for some of the less frequently addressed issues in the related literature, such as the number of restrictions, documented classes and naming conventions. Even though our approach falls in the same category with Ontometric and oQual, it introduces some advances over them. In particular, both Ontometric and oQual require more time and effort to deploy due to their complexity both in terms of their structure and the number of supported evaluation criteria. On the other hand, the method presented at this document provides a simpler and more straightforward evaluation framework. Beyond this the recommended framework supports numerical representation of quantitative evaluation criteria, as opposed to Ontometric, thus making the overall evaluation process more accurate.

2.5 - Semi-automatic approaches

All the above methodologies are manual activities for evaluating ontologies. However, there are some efforts proposing semi-automatic techniques.

In [23] the metrics that BioPortal calculates for the ontologies in its repository are presented. In general, BioPortal calculates the metrics when the ontology is uploaded and stores it as part of the Ontology Metadata. There are two groups of metrics: *statistical* such as number of classes, properties, axioms, siblings and maximum depth and *quality-assurance* metrics that may give some indication of the quality of the ontology and help ontology authors improve the quality. These metrics can be accessed through the BioPortal user interface and through dedicated REST services. A drawback of this approach is that these quality-assurance metrics cannot be calculated in ontologies with more than 200 classes. Ontologies with more than 200 classes in a category will still have the total number of classes in that category counted, but no quality list will be available.

In addition, Ontology Metrics [24] is a web-based tool that validates and displays statistics about a given OWL ontology, including the expressivity of the language it is written in. Some of the most useful metrics are: number of classes, object properties, data type properties, individuals and annotation properties. It accepts ontologies written in RDF/XML, OWL/XML, OWL Functional Syntax, Manchester OWL Syntax, OBO Syntax, or KRSS Syntax and it is powered by the OWL

API. What can be seen as a drawback about this tool is that it covers only basic metrics and cannot provide information for more complicated hierarchy metrics like maximum depth, number of siblings and parents, number of internal and external nodes as opposed to our approach.

Moreover, LExO (Learning Expressive Ontologies) [25] is a tool for transforming natural language definitions into OWL DL axioms. The core of LExO is a syntactic transformation of natural language sentences into description logic axioms. Given a natural language definition of a class, LExO starts by analyzing the syntactic structure of the input sentence. The resulting dependency tree is then transformed into a set of OWL axioms by means of manually engineered transformation rules. To ensure the consistent evolution of the learned ontology LExO uses an algorithm which identifies minimal parts of inconsistencies and removes them from the ontology. Thus, LExO could be used for resolving inconsistencies in ontologies that are automatically created by lexical resources.

In [27] authors analyzed the learning problem in Description Logics. First they introduced a general framework for different learning methods. Then they analyzed refinement operators as the main method to traverse the space of concepts ordered by subsumption. These operators in combination with the learning algorithm could be used for ontology creation and maintenance. Similarly, in [28] a learning algorithm for description logics, was created, which also makes use of refinement operators—however, not as centrally as in previous approach. The core idea of this algorithm is blame assignment, i.e. to find and remove those parts of a concept responsible for classification errors. Instead of using the classical approach of combining refinement operators with a search heuristic, a different approach is taken therein for solving the learning problem by using approximated MSCs (most specific concepts). However, a problem of these algorithms is that they tend to produce unnecessarily long concepts.

In [30] we have an automated method for checking the usage of an ontology's schema entities (concepts, axioms) by a set of individuals. The novel aspect of the ontology coverage check lies very much in extending the meaning of unpopulated areas from concepts without individuals to axioms without satisfying individuals. Among the existing approaches to evaluation, we can most closely relate the OCC to the approach of data driven ontology evaluation described by Brewster [13]. The OCC shares two common points with Brewster. First the idea that an ontology should match a very concrete domain defined by a text corpus. Second, that the extent to which the data cover the ontology is a measure of appropriateness of the ontology. However, there are various differences. Some of them are due to the fact that Brewster's method is designed for ontology selection instead of for ontology engineering. Another difference is, that the OCC does not use data outside the ontology (like a text corpus for example), but assumes the data to be already available in an ontology format, i.e. as individuals in OWL terminology. On the other hand, the OCC is able to consider axiom usage by the data.

The OntoManager [36] tool provides an easy-to-use management system for administrators, domain

experts, and business analysts, since they are able to use it productively, with a minimum of the training. OntoManager is best applied in domains in which usage information of ontologies is available to identify relevant concepts of an ontology. This occurs mainly in the area of web portals or any other ontology-based application producing so called semantic log files. However, the limitation on usage information does not allow to evaluate an ontology in general. Therefore OntoManager might be used as an additional analysis of an ontology within an existing evaluation process.

Finally, in [26] they proposed the Rapid Ontology Development approach (ROD) during which the user is continuously supported by ontology evaluation and recommendations for progressing to next steps. These recommendations refer to circulatory errors, concept description in natural language and concepts' connection. The applicability of the approach is demonstrated on financial trading domain where a user can build Semantic Web application for financial trading based on ontologies that consumes data from various sources and enable interoperability. However, this approach focuses mostly on ontology modeling and not on evaluation.

3 - A proposed ontology evaluation framework

This section analyzes the most important aspects to be considered during the restructuring phase as part of our evaluation methodology. The analysis that follows is based on the layer-oriented approach that was originally defined in the Ontology Summit 2007 [29]. These layers provide a taxonomy of the identified ontology issues that should be taken into account during the refinement process. Each one of the identified issues or properties indicates a necessary step in the refinement and evaluation process that should be followed in order to improve the ontologies in their original form. The proposed layers or dimensions are distinguished between internal and external ones. Internal dimensions are concerned with the ontologies themselves, their internal organization, naming conventions, representation, and so on. The external measures are related to their take-up and use within user communities, their role as standards, embedding within business practices, and so on.

In particular, the basic internal dimensions or 'layers' are listed below:

1. Lexical/Vocabulary layer – This layer includes all restructuring attributes that are relevant to the syntactic elements of ontologies, such as naming conventions.

2. Structural/Architectural layer – It includes all aspects that characterize the structural attributes of ontologies, i.e., concept and property hierarchy, grouping of similar ontological concepts, that are repeated and removal of unused modules.

3. Representational/Semantic layer – This layer relates to the semantic elements of ontologies, i.e., attributes whose goal is to conceptually describe the structural ontology elements. Disjointness restrictions belong to this layer.

4. Data/Application layer – The fourth internal layer covers attributes relevant to how an ontology applies to a given domain. Domain range definition of properties is listed as an attribute of this layer.

In addition to the internal layers listed above, there is an external one:

5. Usability layer – It includes quality measures that are required to ensure that the resulted ontologies satisfy a set of usability standards. This layer includes documentation and visualization.

All of the aforementioned layers along with the corresponding criteria, as well as the associated metrics that are defined later on in this Section are summarized in Table 1. In the following subsections we present in detail each one of the restructuring criteria that appear in Table 1.

Table 1: Ontological Layers, corresponding restructuring criteria and their metrics.

Layer	Criteria	Metrics
1. Lexical/Vocabulary	Naming Conventions	N1-N3
2. Structural/Architectural	Class Hierarchy/Taxonomy Property Hierarchy/Taxonomy Property Restrictions Structural Modularity	C1-C13 P1-P5, C6-C13 P6-P12 G1,G2
3. Representational/Semantic	Disjointness Restrictions	J
4. Data/Application	Domain and Range	R1,R2
5. Usability	Documentation/Visualization	D1,D2

3.1 - Naming Conventions

Naming conventions refer to the way, in which all elements of an ontology are named. They belong to the lexical/vocabulary layer, because naming is basically part of the syntactic features of ontologies. It deals with the formulation of well-formed terms and definitions, where essential features should be satisfied by all naming conventions (e.g. nominal, verbal, etc.). According to this criterion circularity in definitions should be avoided and junk categories should be eliminated.

There are a number of useful conventions that can be applied in naming that improve reusability: although formally the names given to concepts are arbitrary, in practice the naming can serve a positive role for documentation and understanding an ontology when attempting to assess its relevance for a particular aim. As a consequence, our metrics include particular strategies of improving the quality and consistency of names used within an ontology. As an example, there may be some concepts modeling similar kinds of information. These concepts usually begin with the same prefix and end with a different suffix, or vice versa. However, in practice it is often observed that not always the same prefix/suffix is used. In this case, these concepts should be aligned for reasons of clarification and clearness and follow the same naming conventions (e.g. begin with the same prefix or end with the same suffix).Furthermore, plural/singular forms and the use of camel case or use of the underscore symbol should not be mixed. According to this criterion, our evaluation metrics propose that one common naming convention always be adopted, such as the camel-case and use e.g. only singular form throughout the ontology.

So for example a class with the name Local_support_groups has been renamed to LocalSupportGroup, so that camel-case and singular form are used. The property with the name LongBench has changed to longBench because it is a property name and therefore should begin with a lower case letter. Other examples include renaming Availability_assistance_services_Hearing to AvailabilityAssistanceServiceFor HearingImp, or clear_signs to clearSign, etc.

Whereas providing this kind of clean-up is straightforward, there are also naming conventions that go further in that they can provide more of an indication of the structure of an ontology as a whole; we return to these when we discuss the class hierarchy below.

Evaluation Metrics

In order to assess in a measurable way how well the naming conventions criteria are fulfilled by an existing ontology we introduce the following three metrics.

N1: Classes with the same naming conventions. This metric is equal to the percentage of the majority of classes that adopt the same naming convention schema, such as camel-case notation, singular form of words and upper case letter. The value of this parameter ranges from 0%, when none of the classes adopt any naming convention standard, to 100% where all classes adopt the same standard. The value of this parameter indicates the extent to which the ontology adopts a common naming standard.

N2: Object properties with the same naming conventions. This metric is the same as the previous one but it applies on object properties rather than classes and takes into account property names that begin with a lower-case letter.

N3: Data-type properties with the same naming conventions. Similarly, this metric is defined as in the previous case but it applies on data-type properties.

3.2 - Class Hierarchy

Concept taxonomy and hierarchy belong to the structural/architectural layer, because the hierarchies that are defined within concepts and properties determine the way in which the ontology will be structured. On the other hand, ontologies are commonly formed as taxonomies that are built around concrete configurations of different hierarchies amongst ontological elements. A flat concept hierarchy, for instance, usually implies that there are too many concepts on the same level. This strongly suggests the existence of unexploited grouping possibilities for concepts with similar semantics, hence these concepts should be grouped together under more general intermediate concepts. Specifically, the problem with flat concept hierarchy is that everything exists everywhere at once and all on the same level. Thus, there is no modularity, openness or depth in these ontologies and there is a growing appreciation that ontologies are evolutionary. However, evolutionary theory demands a clear identification of variation, interaction and selection but a flat ontology can make no sense of this.

Another case is the existence of branches with different structures. This may result in too deep ontologies and unbalanced taxonomies. Finally, the level of generality to which the concepts refer is not always taken into account with sufficient careful, thus resulting in an inappropriate ontology

structure.

All of these issues need to be considered during the ontology design or restructuring phase. For instance, a flat concept hierarchy can be converted to a more arbore scent (tree-like) structure, so as to reduce the number of concepts on the same level. Exploiting the grouping possibilities for concepts of similar kinds results in a better grouping and a more clear reorganized structure of the ontology. A more appropriate structure for ontologies can also be achieved by grouping together on the same hierarchy level all concepts that refer to the same level of generality. Finally, the structure of branches, which are very different than others can change in order to have a more balanced and equally developed hierarchy.

So let's give two examples that illustrate the concept of hierarchy/taxonomy restructuring. If we have an ontology with the classes Bar, Cafe and Restaurant on the same level we can group them together under the more general concept FoodAndDrink that represents food and beverage facilities. Or if we have an ontology with the classes Climate, MeanTemperature and Temperature on the same level we can organize them on a three-level structural schema, according to the semantics of each class (e.g. MeanTemperature and Temperature can be classified as subclasses of Weather, which in turn becomes a subclass of the class Climate).

This then needs to be taken further with respect to the naming conventions. A further convention (perhaps introduce this above after all) is to ensure that the internal structure of the names that are selected match the intended class hierarchy. This can have a dramatic effect both on the intelligibility of an ontology and on its appropriate reuse. In general it should be ascertained for each complex concept label whether it exhibits the linguistic structure of Modifier+Head. The Head should correspond to the class-subclass concept hierarchy. Modifiers may then distinguish subclasses. In the present case, we have a complex head: “Food” and “Drink”. Under their usual interpretations these names would suggest that we are in a portion of the concept hierarchy to do with consumable foodstuffs. But this turns out to be incorrect since we are in the portion of the hierarchy concerned with types of places for consuming these foodstuffs.

Here we therefore suggest a further correction of the name: the less appropriate “FoodAndDrink” needs to be replaced with something along the lines of “FoodAndDrinkFacility”. This immediately renders automatically generated documentation more intelligible and incorporates more of the designers intentions directly in the naming.

A similar evaluation can apply in the second case here: it can be asked whether “Temperature” is a subclass of “Weather” –presumably not, so actually we probably have properties here rather than subclasses.

Evaluation Metrics

The concept hierarchy indicates how well a specified taxonomy is structured. The measurable criteria that are used in order to assess this feature are associated with the number of classes, average number of parent and sibling nodes, as well as various metrics about the characteristics of the tree taxonomy, such as the tree depth, the internal and external paths, and so forth. The total list of these criteria follows.

C1: Total Number of Classes. It is defined as the number of named classes in the ontology.

C2: Number of Primitive Classes. This metric equals the number of classes in the ontology that have only necessary conditions. When necessary conditions are defined for a class, any instance of this class should necessarily fulfill these conditions. However, if any instance fulfills these conditions, this does not necessarily imply that it is also a member of this class.

$$C2 = C1 – C3$$

C3: Number of Defined Classes. This is equal to the number of classes in the ontology that have at least one set of necessary and sufficient conditions. When necessary and sufficient conditions apply to a class, any member, i.e., instance of this class should necessarily fulfill these conditions, and vice versa, if any instance fulfills these conditions then it is certainly a member of this class.

C4: Average Number of Parents. This metric expresses the average number of parent classes, or “super-classes” based on each class in the taxonomy. The greater the value of this metric is, the denser the structure of the ontology becomes.

$$C4 = (\text{The sum of super-classes of all classes}) / C1$$

C5: Maximum Number of Parents. Similarly to the previous metric, this one is equal to the maximum number of super-classes measured over all ontology classes. This is a structure related metric that expresses the maximum number of Isa hierarchy associations that are defined per class.

C6: Average Number of Siblings. This metric is the average number of sibling classes, i.e., classes that share the same parent of all ontology classes. This metric expresses the average number of child nodes per hierarchical level and parent class. As the value of C6 increases, the ontology becomes denser, and the number of child nodes increases per parent node.

$$C6 = (\text{The sum of adjacent sub-classes of all classes}) / (\text{Number of parent classes})$$

C7: Maximum Number of Siblings. This metric displays the maximum number of classes that share the same parent node in the ontology. This is also a metric of how dense an ontology is in terms of its structure. A large value for C6 indicates a dense ontology with a great number of child nodes per parent node.

C8: Max Depth. Given an ontology tree, this metric computes the maximum depth of the tree structure, namely the number of nodes along the longest path from the root node down to the farthest leaf node. This metric indicates the number of structure levels within the ontology. A big value for C8 indicates that the taxonomy consists of many hierarchy levels.

C9: Total Number of Nodes. This is the total number of nodes in the ontology tree structure. This is a metric about how dense the ontology structure is.

$$C9 = C1 + 1$$

C10: Total Number of Roots. The total number of nodes that belong to the topmost level in the ontology tree hierarchy, i.e., the number of nodes with no parents. This indicates the number of independent classes that are defined within the same taxonomy. It is a measure of ontology modularity.

C11: Total Number of Internal Nodes (Parents). This is equal to the total number of nodes in the ontology tree. Only nodes with child nodes are taken into account. This metric expresses how dense is the ontology structure.

$$C11 = C1 - C13$$

C12: Total Number of Children. This is equal to the total number of child nodes in the taxonomy, i.e., nodes with at least one parent node. This metric also expresses the density of the tree structure.

C13: Total Number of External Nodes (Leaf). This is defined as the total number of nodes in the ontology tree structure that do not have any child nodes. Root nodes are also taken into account for the calculation of this metric. Again, this is a taxonomy-density metric.

3.3 - Property Hierarchy

This and the next refinement criterion, i.e., Property Restrictions, belong to the structural/architectural layer of the ontology authoring process because hierarchy applies to properties in a similar way to that of concepts. Property structure may be quantitatively evaluated by similar metrics as in Section 3.2 such as the size, the depth/breadth of hierarchy, density and complexity of the hierarchy of object and data properties. Issues that are addressed by this criterion include the lack of well-structured properties in ontologies when there is a clear hierarchical relationship between different properties that share common characteristics. The need for adding hierarchical relationships between properties occurs when properties are poorly gathered into conceptual groups of similar properties. In this case a restructuring process is often necessary by exploiting grouping possibilities for properties of equal domains/ranges or their functions. By introducing one or more levels of hierarchy between these properties we achieve a more efficient representation of the involved properties that also results in the reduction of redundant information within the definition of each property. On top of this, the application of the restructuring process to ontology properties can reduce the number of properties on the same level and produce a more hierarchical structure over properties. This implies a more concrete and understandable ontology structure.

Figure 1 illustrates an example of how to apply hierarchy in properties by grouping together properties that are defined in the same context. The properties dayValue and nightValue, which are both properties of MeanTemperature, have been subsumed as generalInfoOfMeanTemperature. On the other hand properties minValue and maxValue, which are both properties of Temperature, have been subsumed under property generalInfoOfTemperature. Similarly, properties generalInfoOfMeanTemperature and generalInfoOfTemperature, together with the new added properties cloudy, rainy, snowy and sunny were subsumed as generalInfoOfWeather, which finally becomes a subproperty of generalInfoOfClimate.

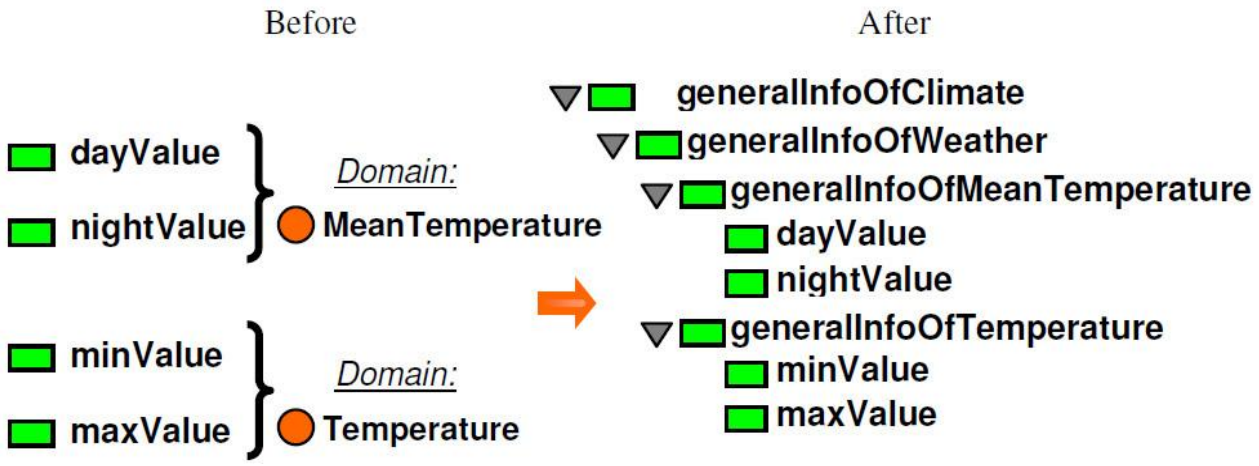


Figure 1: Example of restructuring properties based on equal domains.

After the application of the refinement process properties dayValue and nightValue become subproperties of generalInfoOfMeanTemperature, while minValue and maxValue become subproperties of generalInfoOfTemperature. Finally, the properties generalInfoOfMeanTemperature and generalInfoOfTemperature become subproperties of generalInfoOfWeather.

Evaluation Metrics

General property metrics are used to measure the total number of properties in the taxonomy, as well as the total number of properties of each type (i.e., object, data-type and annotation properties). In particular, the following metrics are defined.

P1: Total Number of Properties. This metric is equal to the total number of properties in the ontology (including object, data-type, and annotation properties). It holds that P1 = P2 + P3 + P4; metrics P2, P3 and P4 are described below.

P2: Number of Object Properties. This is equal to the number of object properties in the

ontology. Object properties provide associations between individuals of the same or different classes in the ontology.

P3: Number of Data-type Properties. Similarly, this metric is defined as the number of data type properties that associate individuals to XML-schema data types or RDF literals.

P4: Number of Annotation Properties. This metric counts the number of annotation properties. These properties are used for documentation purposes, such as to add metadata to classes, individuals and properties.

P5: Properties with an inverse specified. This provides the number of properties for which an inverse property is specified. For example, in the general MIOntos ontology the inverse of the property hasUseCase is the object property hasService.

The structural characteristics of properties are treated in a way similar to that adopted for classes, as analyzed in the Class Hierarchy/Evaluation Metrics section. Metrics in this category address the measurability of features such as the depth and average number of siblings, total number of internal and external nodes, etc. These measurable characteristics represent the extent to which the properties in a taxonomy are structured. For the assessment of the property-related structural characteristics of an ontology we use metrics C6 to C13 that have been introduced for the assessment of concept hierarchy, but which are also applicable in the case of properties. Here we use the same metrics, applying them to properties rather than concepts. This is logical as, with SHOIQ, SHIQ, etc., the various properties can be also structured in the same way as classes. For example C8, maximum depth, is equal to the maximum depth of all property trees, namely the number of nodes along the longest path from the root node down to the farthest leaf node. This metric indicates the number of structure levels within the properties taxonomy.

3.4 - Property Restrictions

We can also use properties in order to create restrictions. This feature is common in the Web Ontology Language3 (OWL). As the name suggests, restrictions are used to impose various restrictions on individuals that belong to a class. Restrictions in OWL fall into three main categories:

- Quantifier Restrictions: AllValues From , SomeValues From.
- HasValue Restrictions.
- Cardinality Restrictions.

Quantifier and hasValue constraints comprise restrictions on the kinds of values that a property can take, whereas cardinality restrictions are applied on the number of values that a property may take. Property restrictions can easily be evaluated by the number of various restrictions that exist in an ontology.

The total time for checking ontology consistency depends on the size of the initial ontology but also on the use of these restrictions. Constructs like SomeValuesFrom, MinCardinality, and

MaxCardinality will cause the consistency algorithm to create new nodes in the ontology. Applying this algorithm to new nodes will require more processing time. Thus, there have to be always a good reason to include restrictions in the ontology. Otherwise, unnecessary restrictions would always result in poor performance of consistency checking mechanisms.

Evaluation Metrics

The appropriate metrics used in order to enable measurable evaluation of several restrictions-related features, such as the number of cardinality, existential, universal restrictions, etc. are as follows. These indicate the extent to which the properties in a taxonomy are imposed to various types of restrictions. In particular, we define the following metrics.

P6: Total Number of Restrictions. In OWL, properties are used to create restrictions. This metric is defined as the number of various restrictions that are imposed to individuals (instances) of a class. Restrictions in OWL fall into four main categories: existential, universal, cardinality and hasValue restrictions. Based on these, the following additional metrics P7 to P12 are defined.

P7: Number of Existential Restrictions. This metric is equal to the total number of restrictions applied on individuals with at least one property from a specific range. For example, the restriction "hasGeographicalAvailability some GeographicalAvailability" refers to all individuals that have at least one hasGeographicalAvailability property with range defined by the class GeographicalAvailability.

P8: Number of Universal Restrictions. This is defined as the number of restrictions that are imposed on properties with exactly one range. For example, the universal restriction "hasUserGroup only CognitiveImp" states that the hasUserGroup property for all individuals has exactly one range, that is defined by the class CognitiveImp, thus no other range is allowed for properties.

P9: Cardinality Restrictions. In OWL, we can describe the class of individuals that have at least, at most or exactly a specified number of relationships with other individuals or data-type values. The restrictions that describe these classes are known as cardinality restrictions. Metric P9. is equal to the number of such cardinality restrictions that allow an individual to participate in a fixed number of relationships.

P10: MinCardinality Restrictions. This is equal to the number of restrictions that impose a minimum number of relationships in which an individual is allowed to participate.

P11: MaxCardinality Restrictions. This is equal to the number of restrictions that impose a maximum number of relationships, in which an individual is allowed to participate.

P12: HasValue Restrictions. This metric counts the number of hasValue restrictions that define an anonymous class of individuals as a range for a specific property. The hasValue restriction associates a specific property to a tangible entity (i.e., a string) that is assigned as a value to the property.

The occurrence of restrictions in an ontology indicates that appropriate care has been taken by the

ontology designer on the concrete definition of ontology properties. For this reason, removing such restrictions without knowing the motivation behind their insertion might not always be safe. As a consequence, potentially useful knowledge could be removed resulting in a poor ontology. Hence the number of various types of restrictions as provided by the aforementioned metrics can be seen as useful indicators for improving ontology design, whenever performance issues should be taken into account. Hence, it will be up to the ontology designer to decide about removing any restrictions, if required, as a trade-off for better performance.

3.5 - Similar Concepts

Also in the architectural layer, we define a criterion about grouping similar concepts that appear in ontologies. This criterion is classified in the architectural layer because it deals with modularization issues, such as what modules are defined in the ontology, how they are defined, whether they can be imported/exported/reused and so on. According to this criterion if lexicographically similar ontological concepts are repeated frequently throughout the structure, they can be possibly combined to one module and reused whenever necessary. Hence, duplicate concepts can be defined only once and their use can be extended within other definitions.

The implication of grouping similar ontological concepts in order to avoid their repetition is to render maintenance of the specified modules easier, e.g. it becomes a trivial task for ontology authors to add or remove something in the ontology or to keep track of the naming issues in general, because naming is preserved and this results in less typing errors. In any case, the definition of modules depends on the language to be used, what is intended to represent, and the applicability of reusing the modules.

Evaluation Metrics

The potential for modularity of ontological concepts can be evaluated directly from metrics G1, G2 that are defined below.

G1: Total Number of Similar Classes. This metric provides the total number of similar classes in the ontologies, thus expressing the lexicographic duplicates that exist on them.

G2: Total Number of Similar Properties. It is equal to the total number of similar classes. This metric indicates the number of lexicographic duplicates with respect to the properties in the ontology.

3.6 - Documentation/Visualization

The documentation and visualization criterion belongs to the representational/semantic ontological layer because it encompasses issues such as how the ontology is represented to the outside world

and how it is described in terms of the semantics of its elements. In particular, this criterion addresses documentation and term governance, among others. It involves the activity of enriching the ontology with additional information, such as free text comments or annotations, metadata, implementation code and so on, as well as the collection of documents and explanatory comments generated during the entire ontology building process. In general, this aspect refers to anything that could be helpful to make the ontology more readable to its users.

Based on experience, it seems that documentation and visualization concerns are usually left as a final task by the ontology authors. Thus, ontologies are usually poorly documented, with few or almost no comments. This results in ontologies that, even if consistent in terms of their syntax and semantics, are difficult to use and understand, especially by users not involved in their design who aim to apply or reuse them. In this case as this criterion dictates, the documentation and visualization aspects of an ontology should be improved and comments should be added for a better description and clarification of the various ontology parts. After providing sufficient documentation to an ontology, it will become easier for this to be applied, reused, and consumed by other applications.

Evaluation Metrics

The goal of the documentation/visualization metrics is to assess the amount of information that is included in the ontology for documentation purposes. This information may be included in the various elements in the ontology as free text comments, annotations, or metadata that facilitate the understanding and reuse of the ontology elements by third-party practitioners. We define the following metrics:

D1: Percentage of Documented Classes. This metric provides the total number of documented classes as a percentage of the total number of classes defined in the ontology and it indicates the extent to which the classes of an ontology are documented. The closer to 100% the value of D1 becomes, the more class-related documentation is included in the ontology.

D2: Percentage of Documented Properties. It is equal to the percentage of the total number of documented properties with respect to the total number of defined properties. Similarly, this metric indicates the extent of documentation regarding the properties in the ontology.

3.7 - Properties Domain and Range

The definition of the domain and range in ontology properties is an ontology design aspect that belongs to data/application layer, as it is related to the representation of data in an ontology and more specifically it affects data accuracy, comprehensiveness, explicitness, clarity and consistency. These characteristics have a partial impact on the quality of an ontology when consumed by applications. The definition of range and domain properties comprises the definition of allowed values that may be assigned to data properties, or the universe of discourse, to which an object

property is valid and can be applied.

If we think of a property as a function f : D → R, then D and R represent the property's domain and range, respectively. For example, the domain of the data-type property hasAge that represents one person's age is all individuals that are instances of the class Person. The range of the same property is the set of all positive integer numbers. Object properties link individuals from their domain to individuals within their range.

Properties with poorly defined or nonexistent domains or ranges should be avoided because they result in inconsistencies that prevent applications from properly consuming ontologies. This shortcoming also occurs when ontologies only define restrictions of concepts, in which the range is defined as a condition, instead of providing ranges for their object properties. In this case the range of properties should be directly specified.

Figure 2 illustrates two examples on an ontology about the definition of range for object properties. The first example shows the data property availabilityOfEspeciallyDesignPlan which was not associated with any range. In the new version of the TL ontology the range was set to Boolean, indicating that the two values true or false may be assigned to the aforementioned property. In the second example, the object property hasWalk was associated initially with the domain NatureAreaWalks, but again no range was defined. Instead, a condition of NatureAreaWalks existed according to which, if any object is associated with any other object via the hasWalk property, then the target objects are instances of the class Walk. Thus, Walk is selected as the range of hasWalk (as long as there were no other fillers that could be used for this property).

Figure 2: Examples of range definition for object properties.

Evaluation Metrics

Object and data-type properties by definition may be associated with a specific domain and range. The domain represents the pool of individuals to which the property is applied, while range represents the pool of potential individuals to which domain individuals are mapped. One relevant issue that concerns poorly designed ontologies is the lack of range and domain definitions on properties. This situation precludes reasoners to perform reasoning on these activities. In order to assess the extent to which range and domain is defined for ontology properties, we introduce the following metrics.

R1: Properties with domain. This metric counts all properties in an ontology for which the domain attribute is defined as a valid non-empty domain.

R2: Properties with range. Similarly, this metric provides the number of properties for which a valid non-empty range is defined.

3.8 - Disjointness Restrictions

Last but not least, disjointness restrictions mainly affect the usability layer of the ontology when it comes to be used as part of an overall application, e.g. when instances are added, forms are created, or queries have to be answered. These restrictions are applied on ontology classes or properties in order to apply limitations to the domain in which they are used. Thus, by properly defining classes and properties their usability is enhanced as their reuse by other applications is more effectively supported.

Although most concepts inside the ontology are usually pairwise disjoint with each other, this condition is sometimes missing for some concepts. On the other hand, for some other concepts disjointness might not hold, but there might be an overlap. In such a case, if for example there may exist an individual that is an instance of two classes, the disjointness restriction should be removed from these two classes.

In general, the issue of disjointness restrictions should be considered more carefully during ontology development or restructuring. That is, for concepts where it is necessary, the missing disjointness condition should be added. Similarly, for some other concepts where an overlap may occur and a specific individual may be an instance of all of them, disjointness does not hold and they should not be declared as pairwise disjoint with each other.

An example is shown in Figure 3 where disjointness for SpectatorStand should not hold for its subclasses SpectatorStandForHearingImp and SpectatorStandForUpperLimbImp, since an individual can be defined that is an instance of both subclasses. Therefore, SpectatorStand subclasses are not pairwise disjoint with each other. By carefully revisiting the disjointness restrictions the usability of the ontology is radically improved when it comes to be used as part of an overall application.

Figure 3: Examples of two cases with and without disjointness conditions.

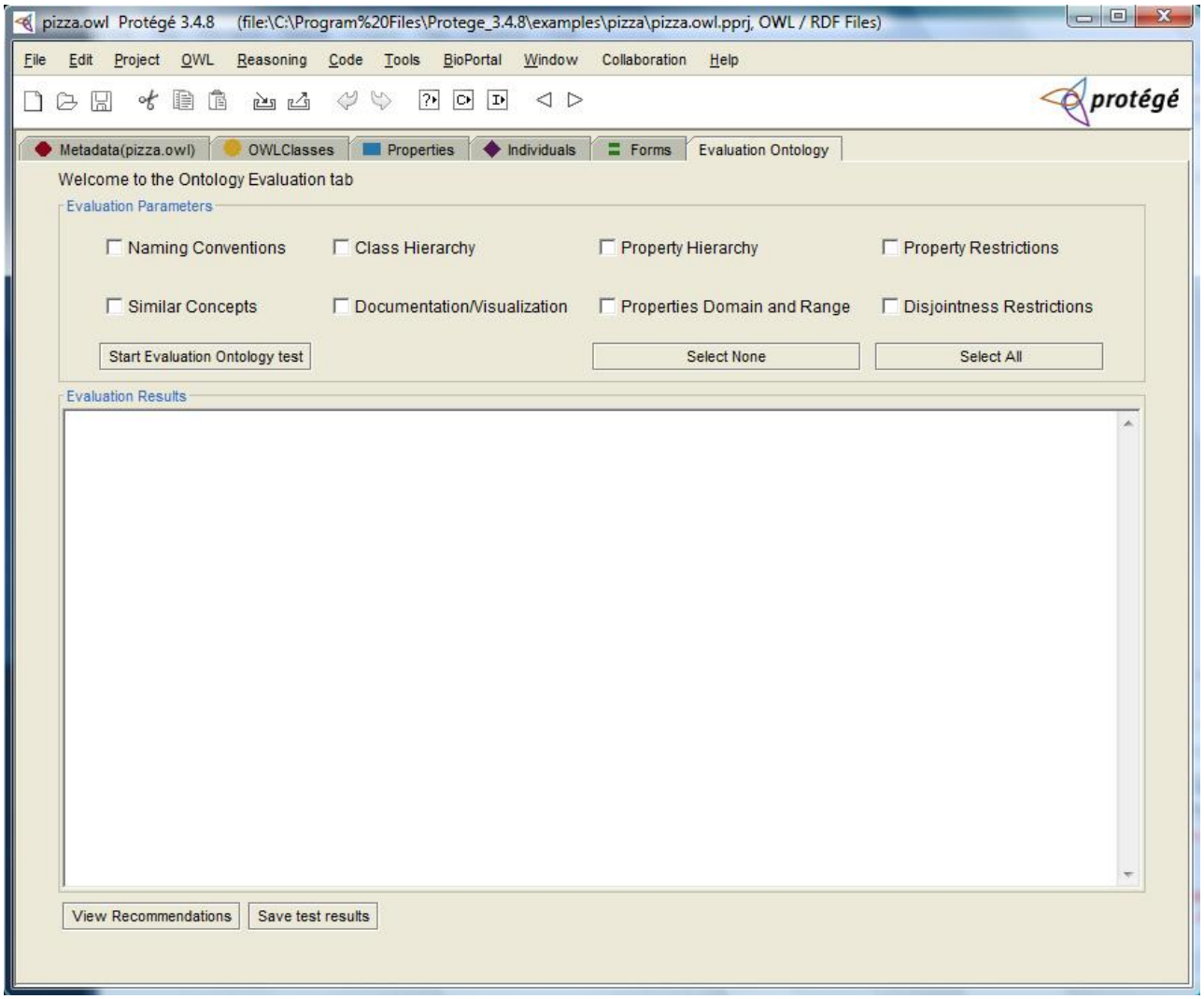
Evaluation Metric

The definition of disjointness restrictions on classes prevents those classes from overlapping with each other, thus creating confusion to reasoners. In order to careful specify the extent to which classes in an ontology are defined as disjoint, we introduce the metric J as the total number of disjointness restrictions on classes. Based on experience, since not all of the classes in an ontology should be disjoint, this metric is used to indicate whether such types of constraints are taken into account or not during the design of an ontology.

4 - The Ontology Evaluation Protégé plug-in

After setting the methodology for the ontology evaluation the next step was to develop an implementation of it. This implementation was made as part of a Protégé (software) plug-in. The plug-in was developed for two different editions of the Protégé, the 3.4.8 and the 4.1. These two editions use different APIs [3] the Jena [4] and the OWL API [5] (respectively for the Protégé 3.4.8 and 4.1) in order to process an OWL [6] ontology. That's the reason for the development of the two different editions. Also, at the end of this document there is the section “Ontology Evaluation Protégé plug-in set up instructions” in which there is a step by step guide regarding the development process of these two different editions of the plug-in.

The next picture shows the interface of the ontology evaluation plug-in for Protégé 3.4.8. (It is identical for the 4.1 edition).



Picture 1: The interface of the ontology evaluation plug-in for Protégé 3.4.8 .

As you can see Ontology Evaluation plug-in consists of two main parts. These are the “Evaluation Parameters” and the “Evaluation Results” which are described below.

Evaluation Parameters

The “Evaluation Parameters” part consists of:

- *8 checkboxes*, each one of them implements the evaluation metrics of the corresponding criteria given by the proposed ontology evaluation methodology in the previous section “3 – A proposed ontology evaluation framework”.
    - “Naming Conventions”
    - “Class Hierarchy”
    - “Property Hierarchy”
    - “Property Restrictions”
    - “Similar Concepts”
    - “Documentation/Visualization”
    - “Properties Domain and Range”
    - “Disjointness Restrictions”
- 3 buttons
    - “Start Evaluation Ontology Test”, starts the procedure and processes the evaluation metrics of the checked checkboxes.
    - “Select None”, deselects all the checkboxes.
    - “Select All”, selects all the checkboxes.

Evaluation Results

The white space that consists the “Evaluation Results” is a text area that depicts the results of the ontology evaluation test after the button “Start Evaluation Ontology Test” is pressed.

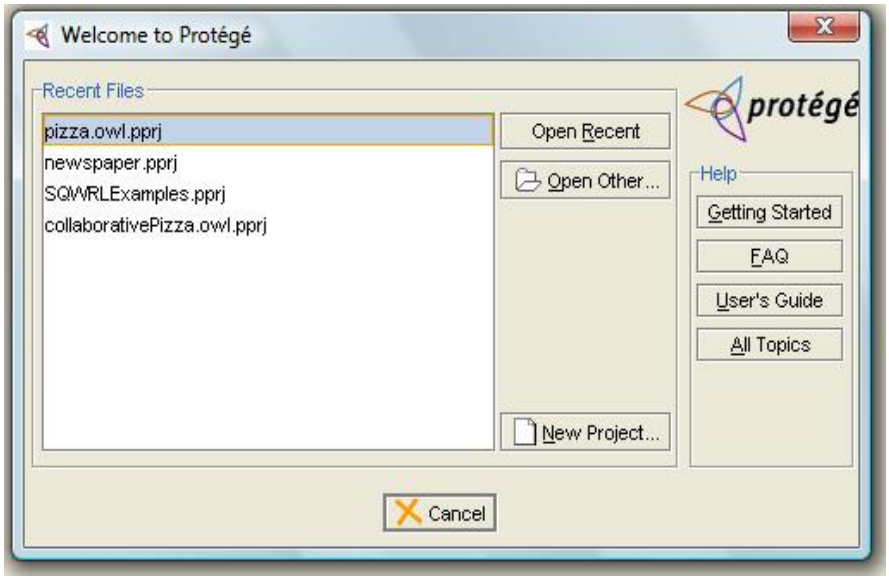
Finally, there are two more buttons on the interface of the plug-in. These are the “Save test results” and the “View Recommendations”. The first one gives the option to the user to save to a text file the results that are depicted on the “Evaluation Results” text area. The second displays on a new pop-up window some indicative recommendations for the user in order to improve the tested ontology. This requires that at least one ontology evaluation test has been performed. The recommendations are presented separately for each one of the ontology parameters that have been selected in the last conducted evaluation test. The rules of the recommendations are defined on an empirical basis, by mainly posting warnings when one or more evaluation metrics appear to have very low values after an evaluation test has been conducted.

Example

In order to provide a more clear view of how the plug-in works let's give an indicative example.

First we start up the Protégé (software).

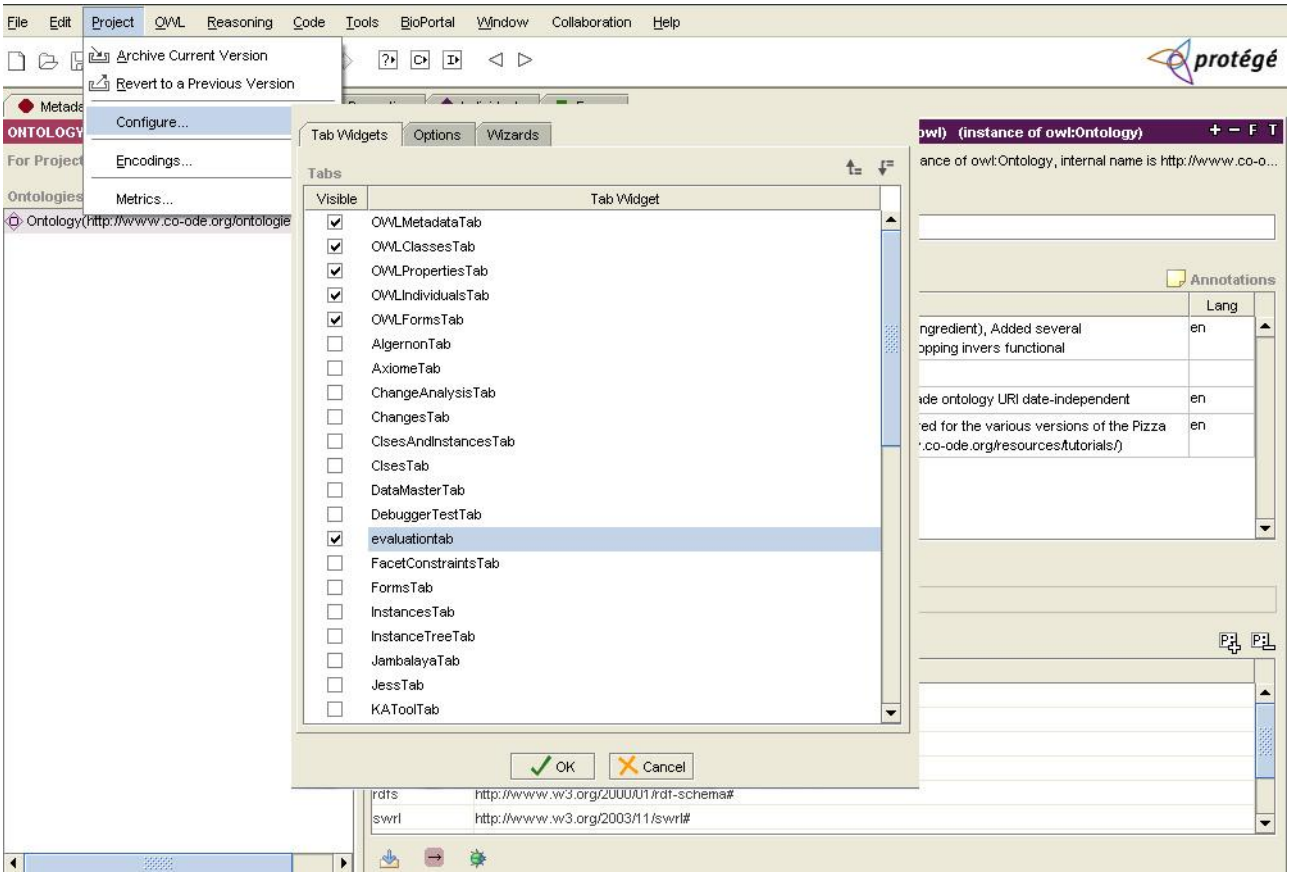
If we use the 3.4.8 edition we see the following window



Picture 2: Welcome to Protégé 3.4.8 window.

Then we click the "Open Other" button and we go to the directory that we have saved our ontology. We select the ontology that we want to load to the Protégé and click ok.

Now the ontology that we want to test is loaded but we've got to display the tab-widget plug-in in order to run the ontology evaluation test. To do this we click Project → Configure and then we select the "evaluationtab" from the list of the Tab Widget plug-ins as seen below.

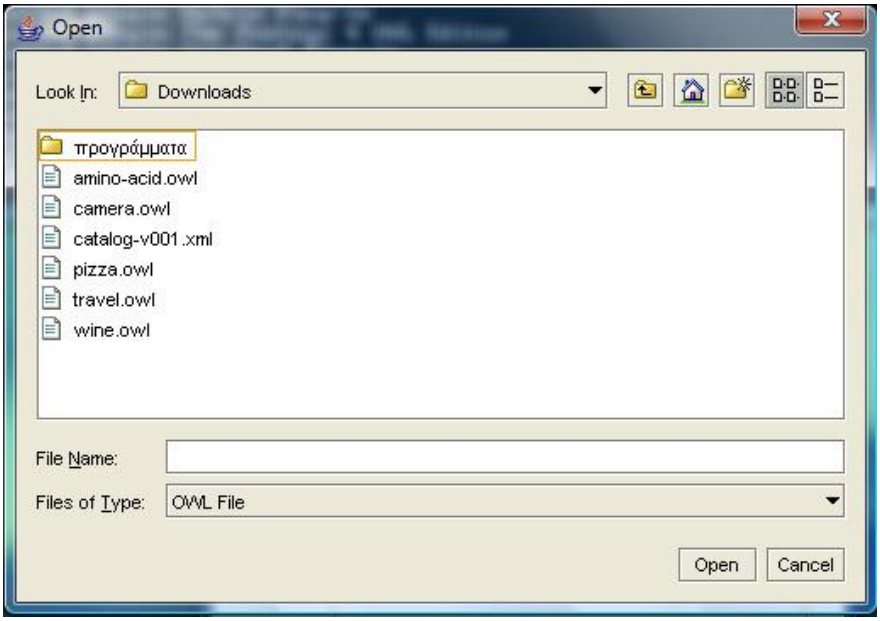


Picture 3: Selecting the evaluationtab on Protégé 3.4.8 .

If we use the 4.1 edition of the Protégé we see the following window when we start it up.

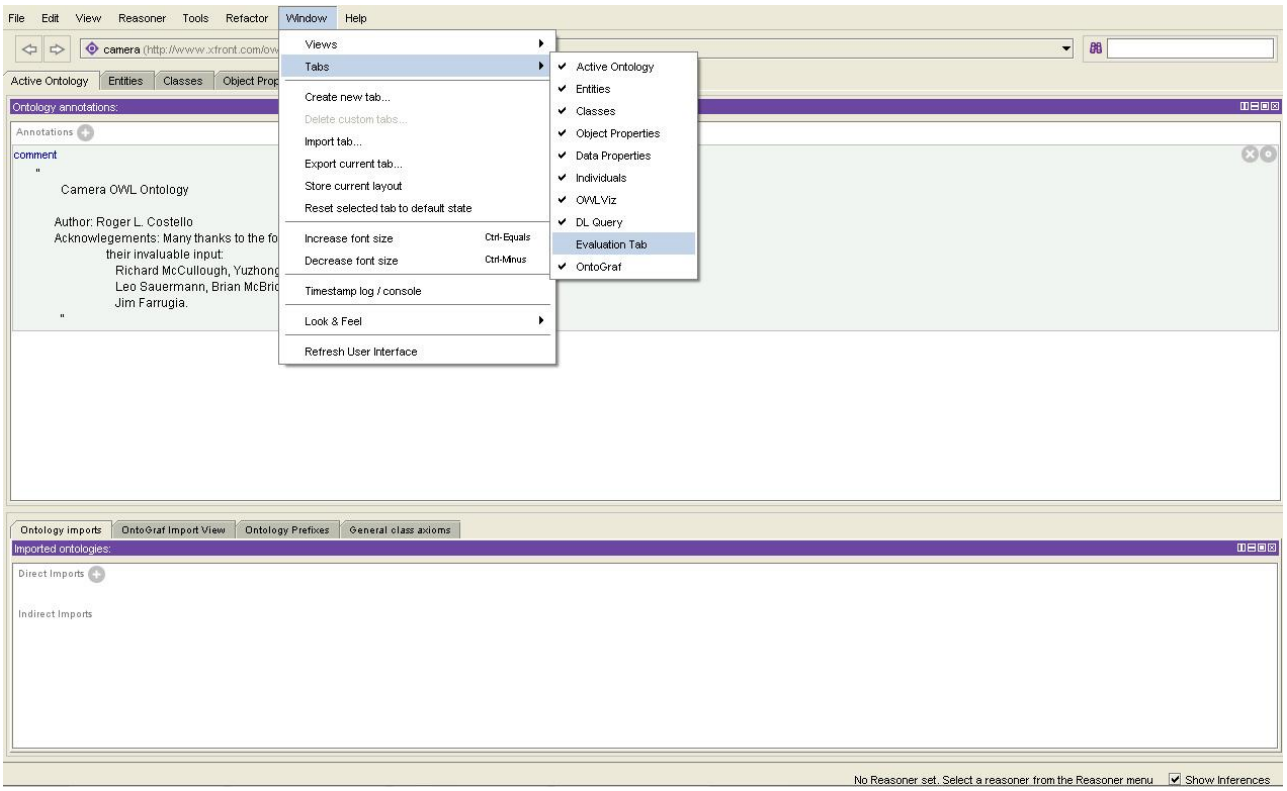
Picture 4: Welcome to Protégé 4.1 window.

We select the “Open OWL ontology” option. We go to the directory that we have saved our ontology. We select the ontology that we want to load to the Protégé and click open.



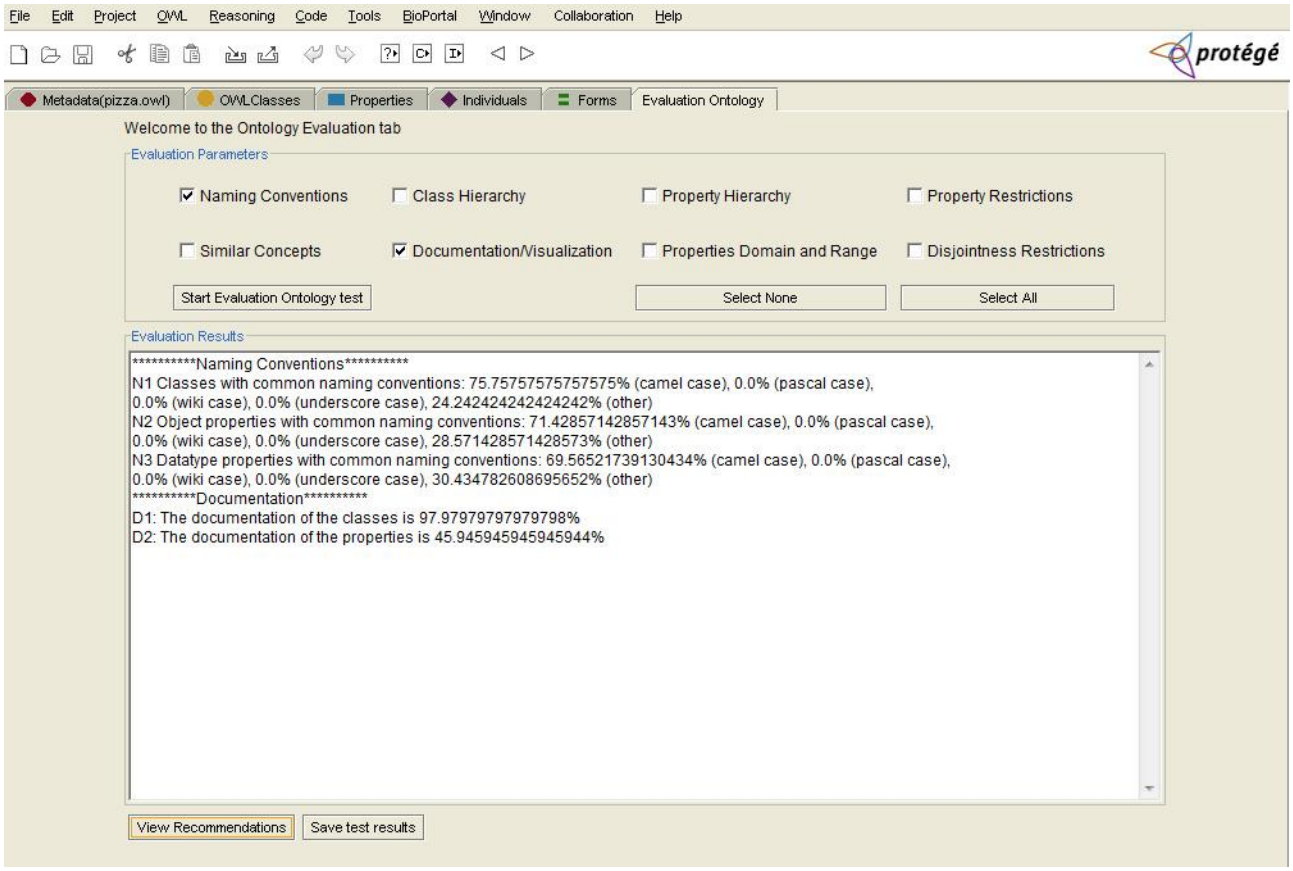
Picture 5: Selecting the ontology to load.

Now the ontology that we want to test is loaded but we've got to display the tab-widget plug-in in order to run the ontology evaluation test. To do this we click Window → Tabs → Evaluation Tab as seen below.



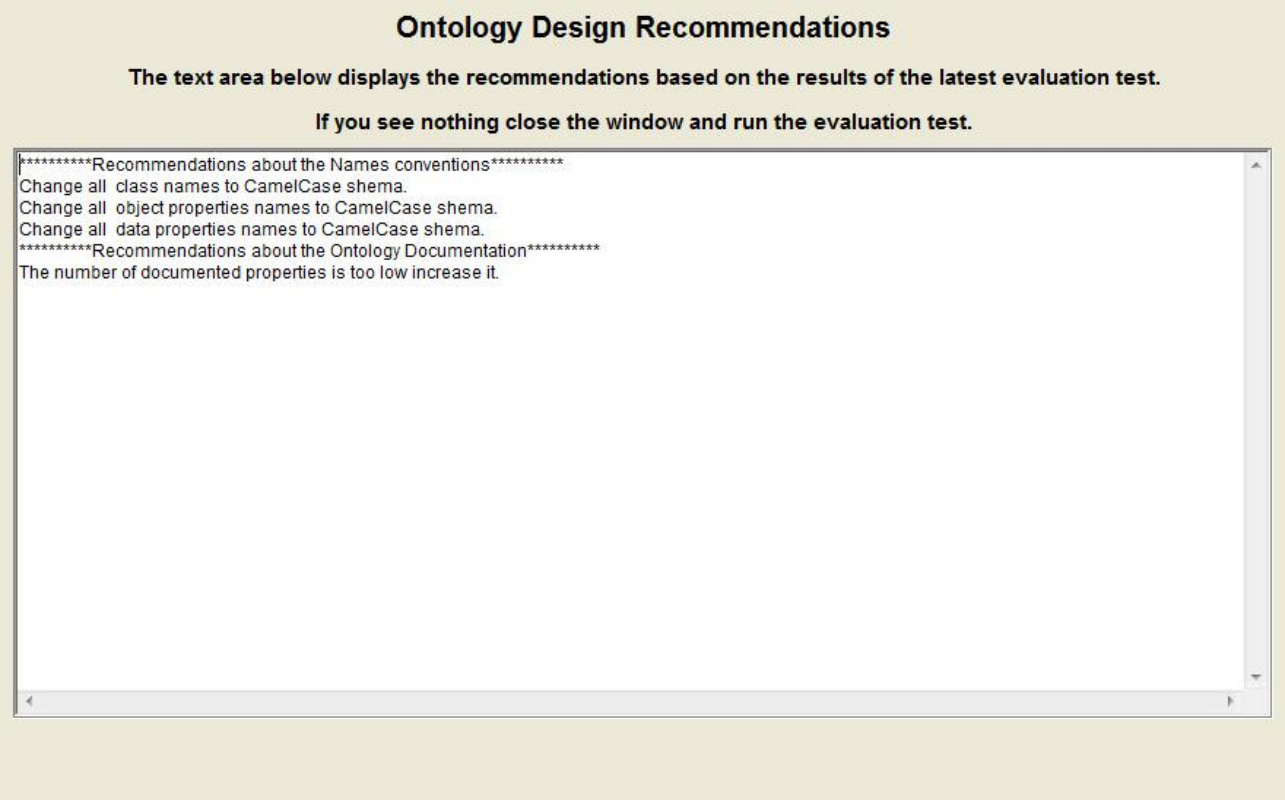
Picture 6: How to display the Evaluation Tab on Protégé 4.1 .

Now let's assume that we want to run the evaluation tests of the "Naming Conventions" and the "Documentation/Visualization" criteria. All we have to do is to check the corresponding checkboxes and click the "Start Evaluation Ontology test" button. Then the results are displayed in the text area "Evaluation Results" as seen below.



Picture 7: Evaluation results example.

If we press the “View Recommendations” button the following window is displayed.



Picture 8: Example of ontology recommendations.

In our case we are prompted to change all the names of the ontology classes, object properties and data properties to the Camel Case schema because this is majority naming schema in the ontology as seen from the results previously. Also we are prompted to increase the number of the documented properties because it is only 45.94% as seen from the results previously. Of course the recommendations that are displayed based on the evaluation test are indicative. By no means it is mandatory for the user to follow them as the core of the ontology evaluation plug-in focuses on the “Evaluation Results”. Each ontology designer takes the “Evaluation Results” and uses them based on his expertise, his knowledge and what is his final goal from the usage of the tested ontology.

5 - Testing the plug-in

In order to show the added value of the ontology evaluation Protégé plug-in, a small indicative scenario was created that is described on the case study below.

Case study

The basic idea for testing the plug-in was to run the Ontology Evaluation plug-in for various ontologies and then to make a comparison of the results and to classify the ontologies wherever it was possible. So for a start we had to choose a set of different ontologies. These ontologies (which can be found on the official Protégé wiki site – url: http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library#OWL_ontologies) are the following:

- *amino-acid.owl*: A small OWL ontology that describes the amino acids and their properties.
- *camera.owl*: An OWL ontology about the individual parts of a photo camera.
- *wine.owl*: An ontology that describes wines (for example the region, the vintage, the wine color, the wine taste).
- *travel.owl*: A tutorial OWL ontology for a Semantic Web of tourism.
- *pizza.owl*: The OWL ontology used in the Protege-OWL Tutorial that describes various kinds of pizzas.

After taking the results from the ontology evaluation plug-in for these ontologies we have chosen three indicative criteria upon which we will classify the ontologies. These criteria are the “Naming Conventions”, “Documentation” and “Similar Concepts”. The next three diagrams classify the ontologies from the best to the worst for each one of the three criteria.

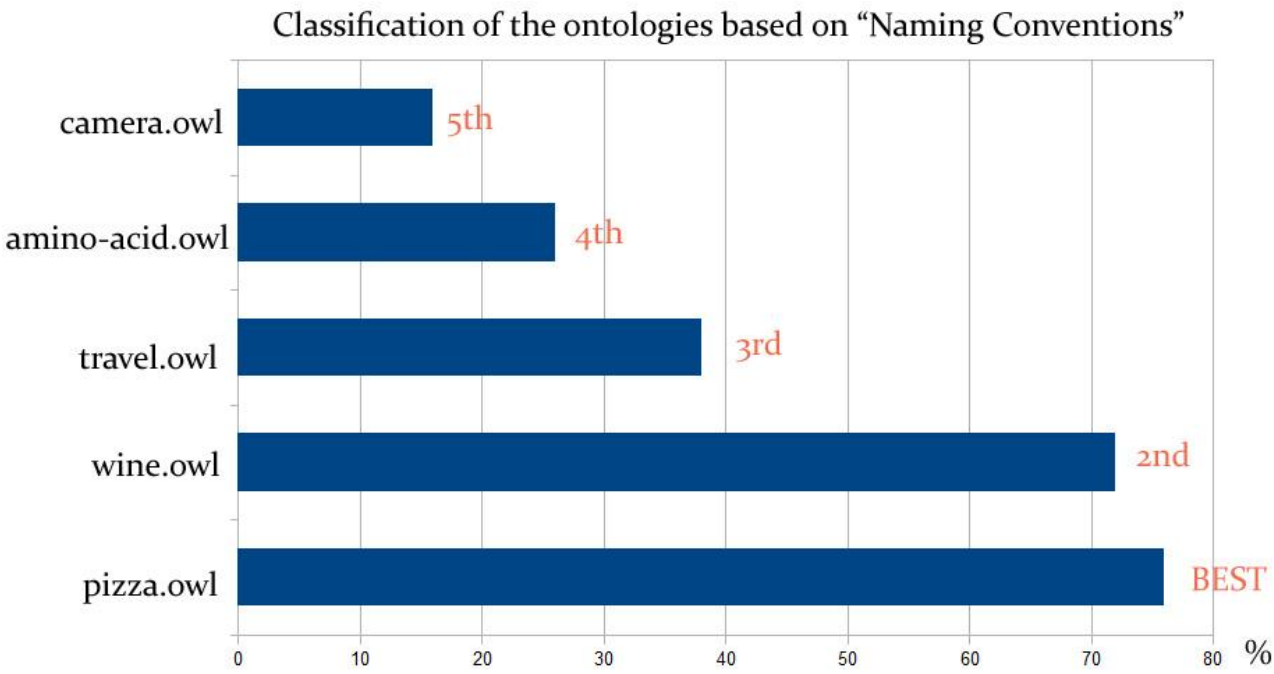


Figure 4: Naming conventions classification.

For the "Naming Conventions" we classified the pizza.owl as the best ontology among the five (seen above) because we wanted the majority of an ontology classes to follow the Camel Case schema for their names.

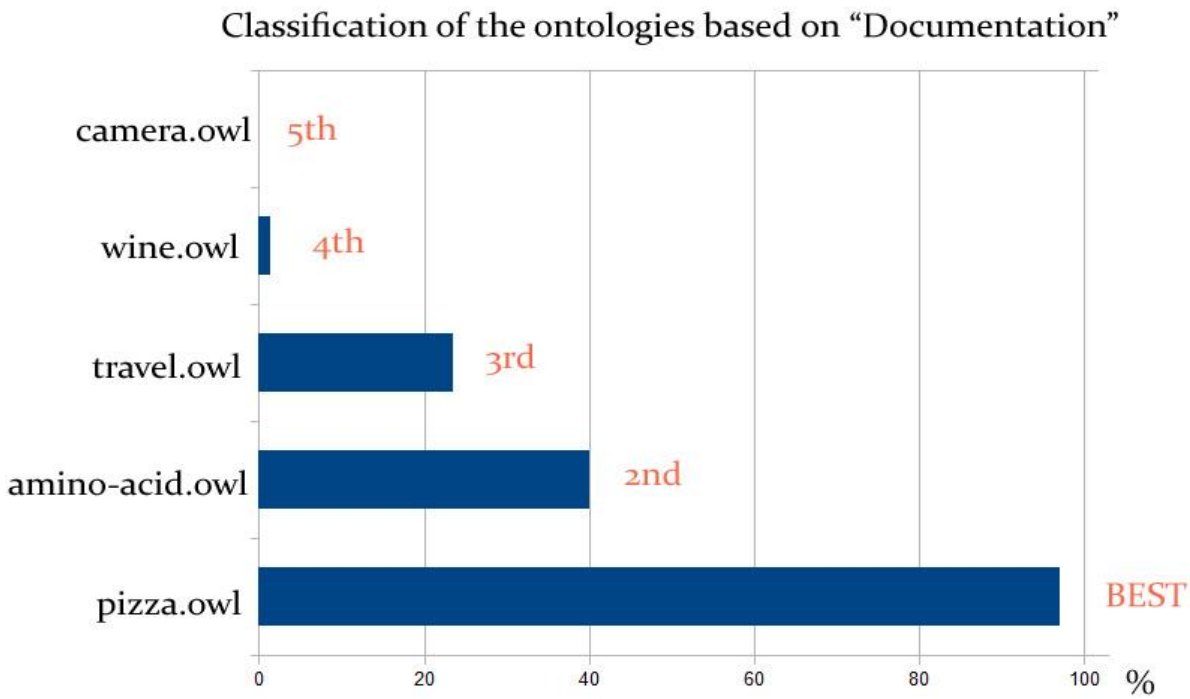


Figure 5: Documentation classification.

For the "Documentation" we classified the pizza.owl as the best ontology among the five (seen above) because it has by far the most of its classes documented.

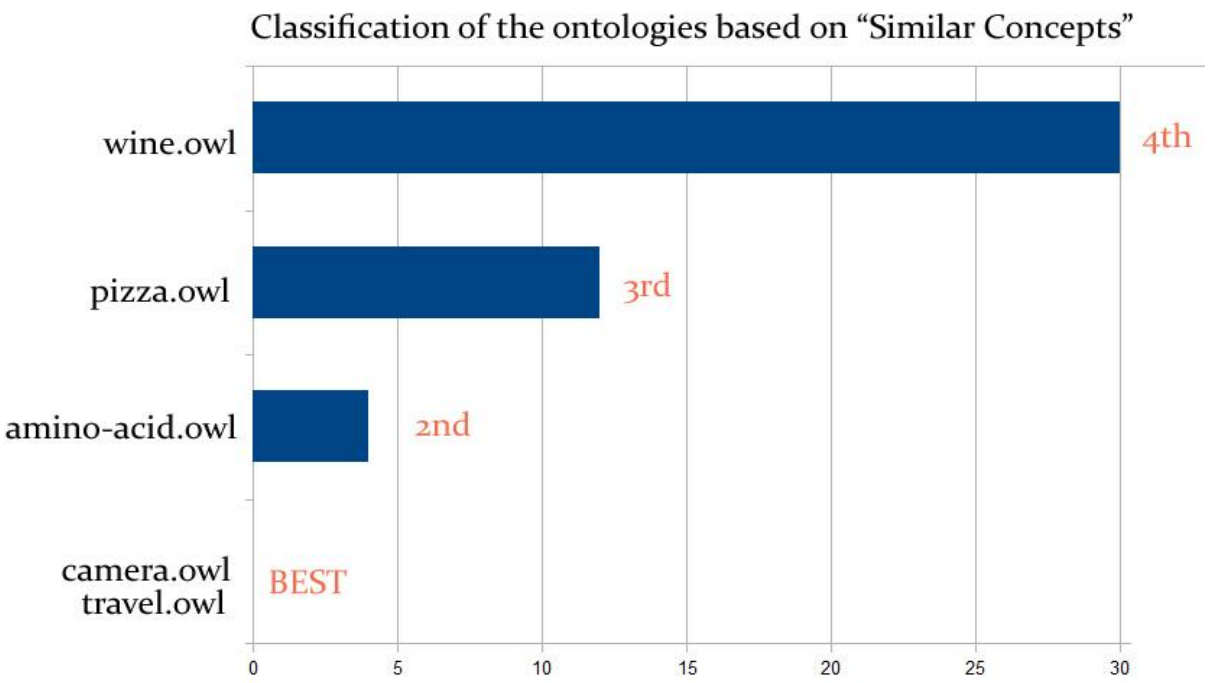


Figure 6: Similar concepts classification.

Finally, for the "Similar Concepts" we classified the camera.owl and travel.owl as the best ontologies among the five (seen above) because we wanted our ontology to have the least duplicate or similar names as far as their lexicographic meaning.

What was shown above was an indicative example of the added value that provides us the ontology evaluation plug-in for the Protégé. Of course we could have chosen any other of the plug-ins criteria in order to classify the ontologies. What was tried to be shown is that with the ontology evaluation Protégé plug-in an ontology designer has a useful tool on his hands so as to evaluate any ontology and to draw a conclusion based on his needs and expectations by taking in zero time and with accuracy the necessary information.

6 - Conclusions and Future Work

In this document it was presented an ontology evaluation methodology whose goal is to provide a set of guidelines and indicate a best-practice approach for ontology re-structuring and refinement. The resulted ontologies are characterized by a better structured taxonomy in terms of their concepts and properties, as a result of eliminating duplicates and definitions of concepts, and reusable ranges of values for several properties. The resulted ontologies, compared to their original versions, are more lightweight, well documented and readable. Thus, the evaluation process that is presented in the paper sufficiently shows that the particular set of criteria that frame our evaluation methodology can form the basis of a formal ontology restructuring process.

By reviewing relevant work and particularly the most well-known ontology evaluation frameworks it is concluded that this methodology fills a gap in the relevant literature. More specifically, to the best of our knowledge, our methodology introduces for the first time such a complete set of measurable criteria that can be applied directly as a tangible means for ontology evaluation. The defined metrics may be used either to compare different ontologies, with respect to some, or all of the involved characteristics. This renders our methodology more complete and applicable compared to the state-of-the-art approaches.

What is expected of the presented methodology is to shape a formal ontology evaluation framework that can be applied in a two-fold way; firstly, as a set of guidelines and best practices for newly created ontologies, and secondly, as a formal ontology framework for existing ontologies. In order to achieve this expectation it was developed a supporting software framework with a set of tools that automate the evaluation metrics process, as much as possible. This software framework was developed as a Protégé tab-widget plug-in in order to fulfill in the best possible way an existing and recognized need for a tangible and efficient ontology evaluation framework capable to be used on a large-scale basis. Finally, the ontology evaluation plug-in has been published and anyone can give it a try by visiting the official Protégé Wiki site (url: http://protegewiki.stanford.edu/wiki/Ontology_Evaluation).

Finally, it is worth mentioning the future plans regarding the development and the support of the "Ontology Evaluation Protégé plug-in". It is going to be studied the possibility of adding more

evaluation criteria on the methodology of the ontology evaluation that was proposed in order to enhance the value of our plug-in. For example a possible metric that may be added in the future on the “Ontology Evaluation Protégé plug-in” is the entropy measure of an ontology graph. The entropy will measure how diverse (uncertain) the structure of an ontology is.

References

[1] T. R. Gruber (1993) *A translation approach to portable ontologies. Knowledge Acquisition, 5(2):199-220,* (PDF) Available at: http://tomgruber.org/writing/ontolingua-kaj-1993.htm (Accessed: 29 May 2013)

[2] PROTEGE OVERVIEW. *what is protégé.* [WWW]. Available at: http://protege.stanford.edu/overview/ (Accessed: 29 May 2013).

[3] *Application programming interface.* [WWW]. Available at: https://en.wikipedia.org/wiki/Application_programming_interface (Accessed: 29 May 2013).

[4] *Apache Jena.* [WWW]. Available at: http://jena.apache.org/ (Accessed: 29 May 2013).

[5] *The OWL API* . [WWW]. Available at: http://owlapi.sourceforge.net/ (Accessed: 29 May 2013).

[6] W3C OWL Working Group (2012). *OWL 2 Web Ontology Language Document Overview (Second Edition).* Available at: http://www.w3.org/TR/owl2-overview/ (Accessed: 29 May 2013).

[7] J. Brank, D. Mladenic, M. Grobelnik, "Gold Standard Based Ontology Evaluation Using Instance Asignment". Proceedings of the EON Workshop, 2006.

[8] R. Porzel and R. Malaka, "A task-based approach for ontology evaluation". ECAI 2004 Workshop Ontology Learning and Population.

[9] C. Welty and N. Guarino, "Supporting ontological analysis of taxonomic relationships". Data and Knowledge Engineering vol. 39, no. 1, pp. 51-74, 2001.

[10] K. Dellschaft and S. Staab, "On How to Perform a Gold Standard Based Evaluation of Ontology Learning". Proceedings of the 5th International Conference on Semantic Web, 2006.

[11] E. Zavitsanos, G. Paliouras and G.A. Vouros, "A Distributional Approach to Evaluating Ontology Learning Methods Using a Gold Standard". 3rd Ontology Learning and Population Workshop, ECAI 2008.

[12] P. Spyns, "EvaLexon: Assessing triples mined from texts". Technical Report 09, STAR Lab, Brussel, 2005.

[13] C. Brewster, H. Alani, S. Dasmahapatra and Y. Wilks, “Data driven ontology evaluation”. Proceedings of International Conference on Language Resources and Evaluation, Lisbon, 2004.

[14] W. Daelemans and M.L. Reinberger, “Shallow Text Understanding for Ontology Content Evaluation”. IEEE Intelligent Systems 1541-1672, 2004.

[15] J. Murdock, C. Buckner and C. Allen, “Evaluating Dynamic Ontologies”. Communications in Computer and Information Science (Lecture Notes). Spencer-Verlag. 2011.

[16] A. Lozano-Tello and A. Gómez-Pérez, “ONTOMETRIC: A Method to Choose the Appropriate Ontology”. Journal of Database Management (2003).

[17] A. Gangemi, C. Catenacci, M. Ciaramita and J. Lehmann, “A theoretical framework for ontology evaluation and validation”. SWAP 2005.

[18] A. Felix, K.A. Taofiki and S. Adetokunbo, “On Algebraic Spectrum of Ontology Evaluation”. In International Journal of Advanced Computer Science and Applications (IJACSA), 2011.

[19] K. Supekar, “A peer review approach for ontology evaluation”. Proceedings 8th International Protégé Conference, Madrid, Spain, July 18-21, 2005.

[20] A. Burton-Jones, V. C. Storey, V. Sugumaran and P. Ahluwalia, “A semiotic metrics suite for assessing the quality of ontologies”. Data and Knowledge Engineering, 2004

[21] B.M Good and J.T. Tennis, “Term based comparison metrics for controlled and uncontrolled indexing languages”. Information Research, 14(1) paper 395, 2009.

[22] J. Tao, L. Ding and D.L McGuinness, “Instance Data Evaluation for Semantic Web-Based Knowledge Management Systems”. In 42st Hawaii International Conference on Systems Science, 2009.

[23] Bioportal NCBO wiki. Ontology Metrics. Available at: http://www.bioontology.org/wiki/index.php/Ontology_Metrics (Accessed: 29 May 2013)

[24] OWL Ontology Metrics. Available at: http://www.w3.org/2001/sw/wiki/Ontology_Metrics (Accessed: 29 May 2013)

[25] J. Völker, P. Hitzler and P. Cimiano, “Acquisition of owl dl axioms from lexical resources”. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, Proceedings of the 4th European Semantic Web Conference (ESWC’07), volume 4519 of Lecture Notes in Computer Science, pages 670–685. Springer, JUN 2007.

[26] D. Lavbic, M. Krisper and M. Bajec, “Continuous evaluation in the process of ontology development”. In the Sixth International Conference on Internet and Web Applications and Services (ICIW 2011)

[27] J. Lehmann and P. Hitzler, “Concept learning in description logics using refinement operators”. Machine Learning journal, 2009.

[28] L. Iannone, I. Palmisano and N. Fanizzi, “An algorithm based on counterfactuals for concept learning in the semantic web”. Applied Intelligence, 26(2), 139–159, 2007.

[29] Gruninger, O. Bodenreider, F. Olken, L. Obrst, P. Yim, “The 2007 Ontology Summit: Ontology, Taxonomy, Folksonomy: Understanding the Distinctions”. Journal of Applied Ontology 3:3, 2008, pp. 191-200.

[30] Viktoria PAMMER, Peter SCHEIR, Stefanie LINDSTAEDT. *Ontology coverage check: support for evaluation in ontology engineering.* (PDF) Available at: http://know-center.tugraz.at/download_extern/papers/fomi2006_camera_ready_version_november_2006.pdf (Accessed: 29 May 2013)

[31] Sourish Dasgupta, Deendayal Dinakarpandian, Yugyung Lee. *A Panoramic Approach to Integrated Evaluation of Ontologies in the Semantic Web.* (PDF) Available at: http://ceur-ws.org/Vol-329/paper04.pdf (Accessed: 29 May 2013)

[32] Alani, H., C. Brewster, and N. Shadbolt. 2006. *Ranking ontologies with aktiverank*. In Proceedings of the 5th International Semantic Web Conference, Athens, GA, 5–9 Nov 2006.

[33] Corcho, O. et al. 2004. *ODEval: A tool for evaluating RDF(S), DAML+OIL, and OWL concept taxonomies.* In Proceedings of the 1st IFIP Conference on Artificial Intelligence Applications and Innovations (AIAI 2004), Toulouse, France, 369–382.

[34] Mostowfi, F., and F. Fotouhi. 2006. *Improving quality of ontology: An ontology transformation approach.* In Proceedings of the 22nd International Conference on Data EngineeringWorkshops (ICDEW’06), Atlanta, GA.

[35] Samir Tartir, I. Budak Arpinar and Amit P. Sheth. *Chapter 5 Ontological Evaluation and Validation.* (PDF) Available at: http://www.cs.uga.edu/~budak/papers/eval.pdf (Accessed: 29 May 2013)

[36] Nenad Stojanovic, Jens Hartmann, Jorge Gonzalez. *The OntoManager – a system for the usage-based ontology management.* (PDF) Available at:

https://km.aifb.kit.edu/ws/LLWA/fgml/final/Stojanovic.pdf (Accessed: 29 May 2013)

[37] Kalfoglou, Y. & Hu, B. (2006). *Issues with evaluating and using publicly available ontologies.* In 4th International Workshop on Evaluation of Ontologies for the Web (EON 2006) at the 15th International World Wide Web Conference, Edinburgh, UK. Retrieved 5 June 2007

[38] Patel, C., Supekar, K., Yugyung, L. & Park, E.K. (2003). *OntoKhoj: a semantic web portal for ontology searching, ranking and classification*. In R. Chiang, A.H.F. Laender & E.P. Lim (eds), Proceedings of the 5th ACM International Workshop on Web Information and Data Management (pp. 58–61). New York: ACM Press.

[39] Maynard, D., Peters, Y. & Li, Y. (2006). *Metrics for evaluation of ontology-based information extraction.* In 4th International Workshop on Evaluation of Ontologies for the Web (EON 2006) at the 15th International World Wide Web Conference (PDF) Available at: http://km.aifb.kit.edu/ws/eon2006/eon2006maynardetal.pdf (Accessed: 29 May 2013)

Annex A

Ontology Evaluation Protégé plug-in set up instructions

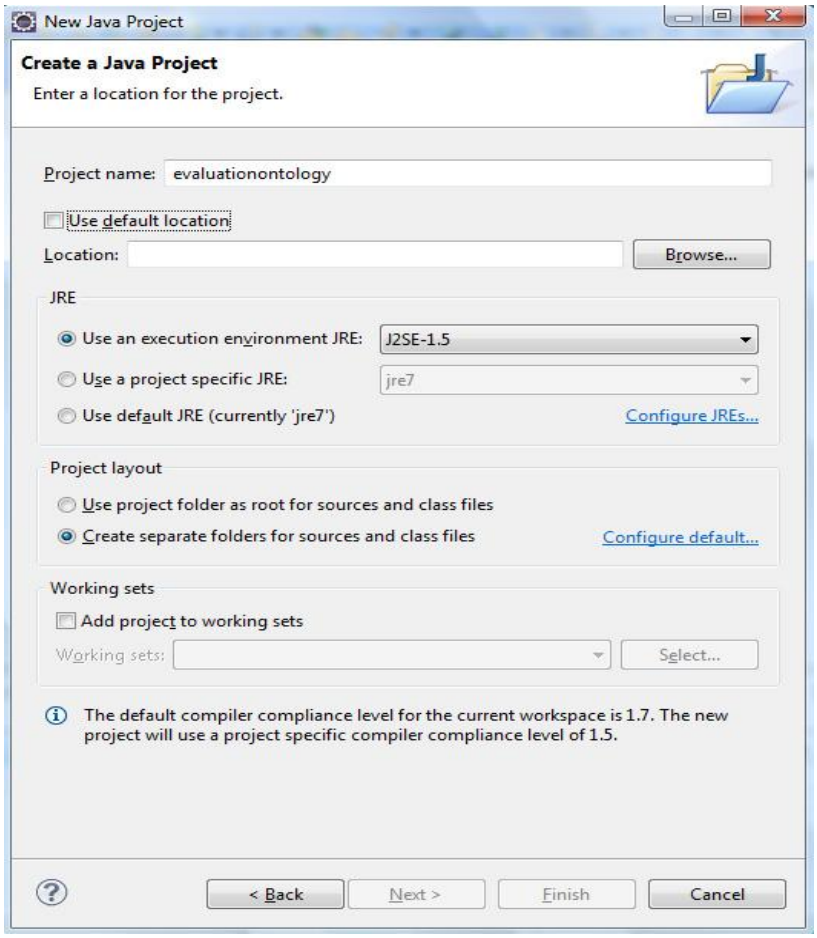
Below you can read a step by step tutorial for building a tab widget plug-in for the Protégé 3.4.8 and the Protégé 4.1. The development was made using the "Eclipse IDE for Java EE Developers" (for more information or to download eclipse click here – url: http://www.eclipse.org/). Of course the code was written using the Java Programming Language.

Tab widget plug-in for Protégé 3.4.8

Setting up Eclipse

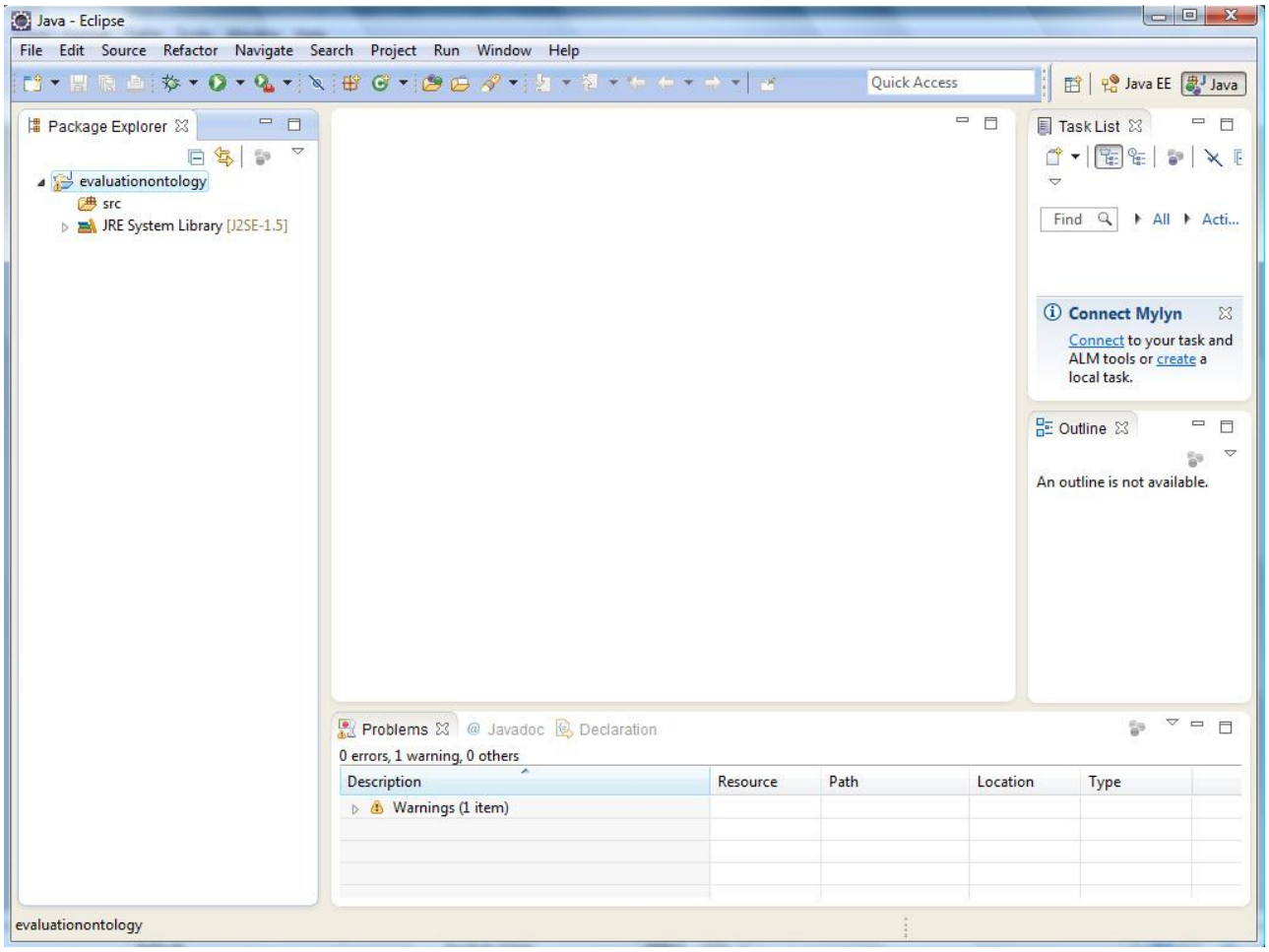
Step 1. Create a new Java Project.

Start Eclipse. Go to File Menu -> New -> Project-> Select Java Project. Click Next. In the next panel choose a project name, say "evaluationontology". Make sure that you choose the J2SE-1.5 as execution environment. The screen should look like below:



Picture 9: New Java Project window.

Click *Finish*. Congratulations! You have created an empty Java project. Your screen should look like:

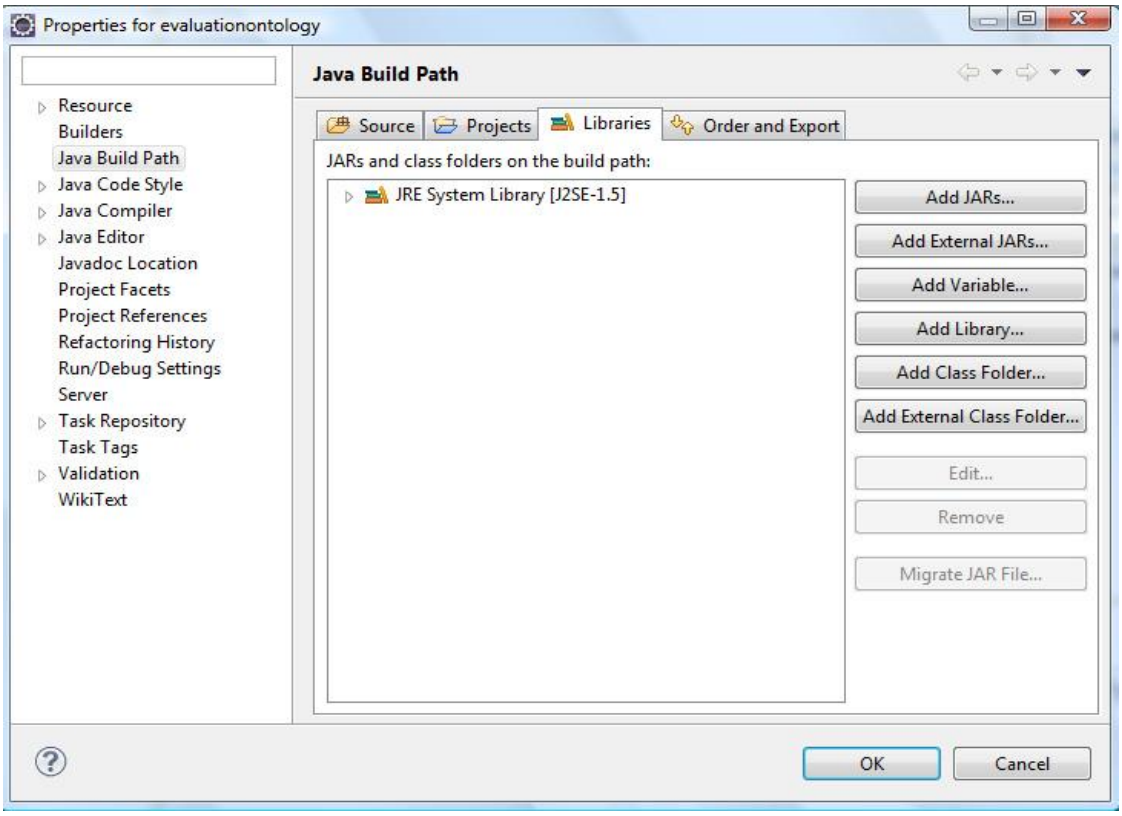


Picture 10: Empty Java Project.

Step 2. Configure the project build path

This step assumes that you have Protege 3.x installed on your computer. This example assumes that Protege was installed in /work/protege/Protege_3.4 (On Windows machines, the default installation directory would be something like C:\Program Files\Protege_3.4)

Select the project name, “evaluationontology”, right click, select Build Path -> Configure Build Path.... Switch to the Libraries tab. You should see something like this:



Picture 11: Configuring the project build path.

Click on Add External JARs... and go to the Protégé installation directory. Select from there the:

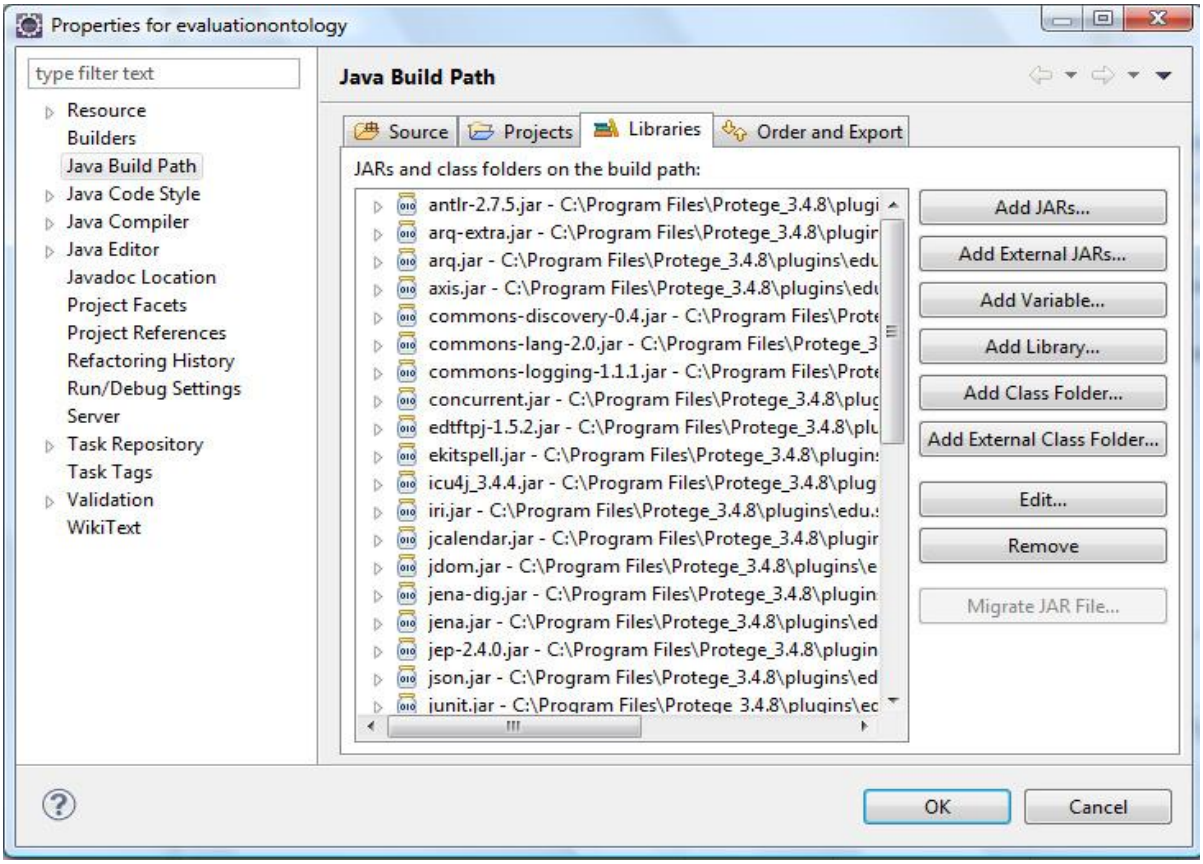
- protege.jar
- looks.jar (version may vary)
- unicode_panel.jar

Click OK.

If your plug-in is for OWL, you will need to include also all the jars in the protege-owl plug-in folder. Repeat the same operation: Click Add External Jars, go to the Protégé installation directory/plugins/edu.stanford.smi.protegex.owl and select all the jar files in that directory.

Note If your plug-in depends on other plug-ins (e.g., on the Change Management plug-in), then you need to add to the build path also all the jars in that plug-in folder.

After adding the Protégé jars (and the protege-owl jars), the Library tab should look like:



Picture 12: Configuring the project build path.

Click *OK*.

Step 3. Create the Java plug-in class

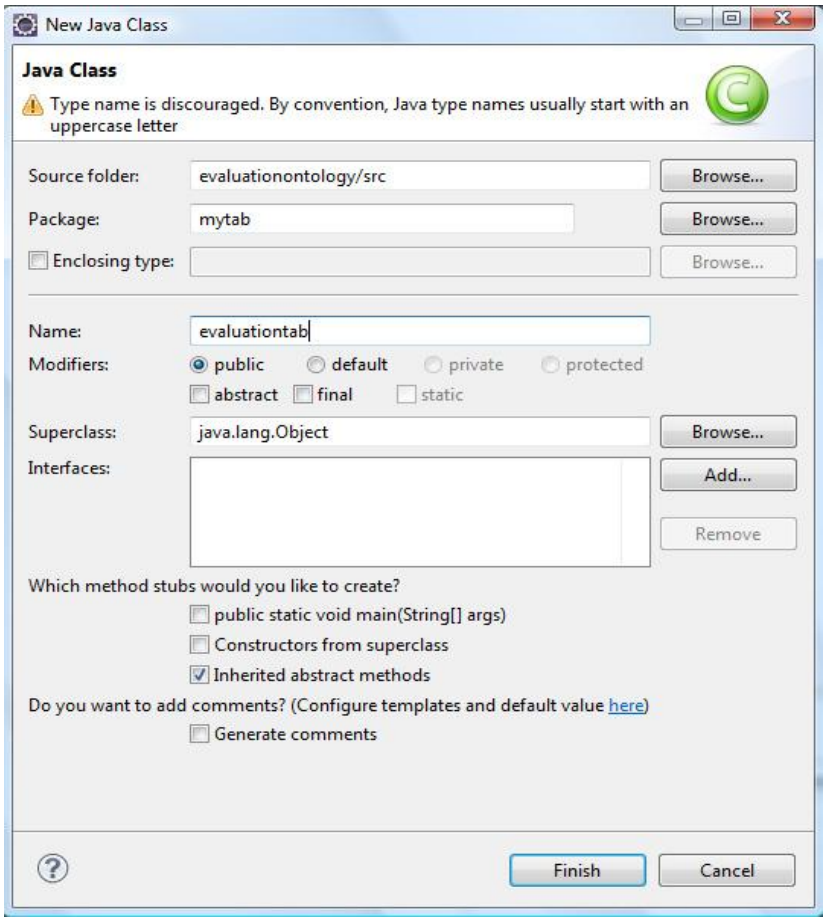
OK, so now Eclipse is set up, all we need is to create a new tab widget. To create a Tab Widget, you need to extend the class edu.stanford.smi.protege.widget.AbstractTabWidget from the protege.jar and implement the method initialize() that is called when the tab is created.

Right-click on src, select New -> Class. In the new class panel:

Write in the Name field: evaluationtab

Write in the Package field: mytab

This is what you should see:

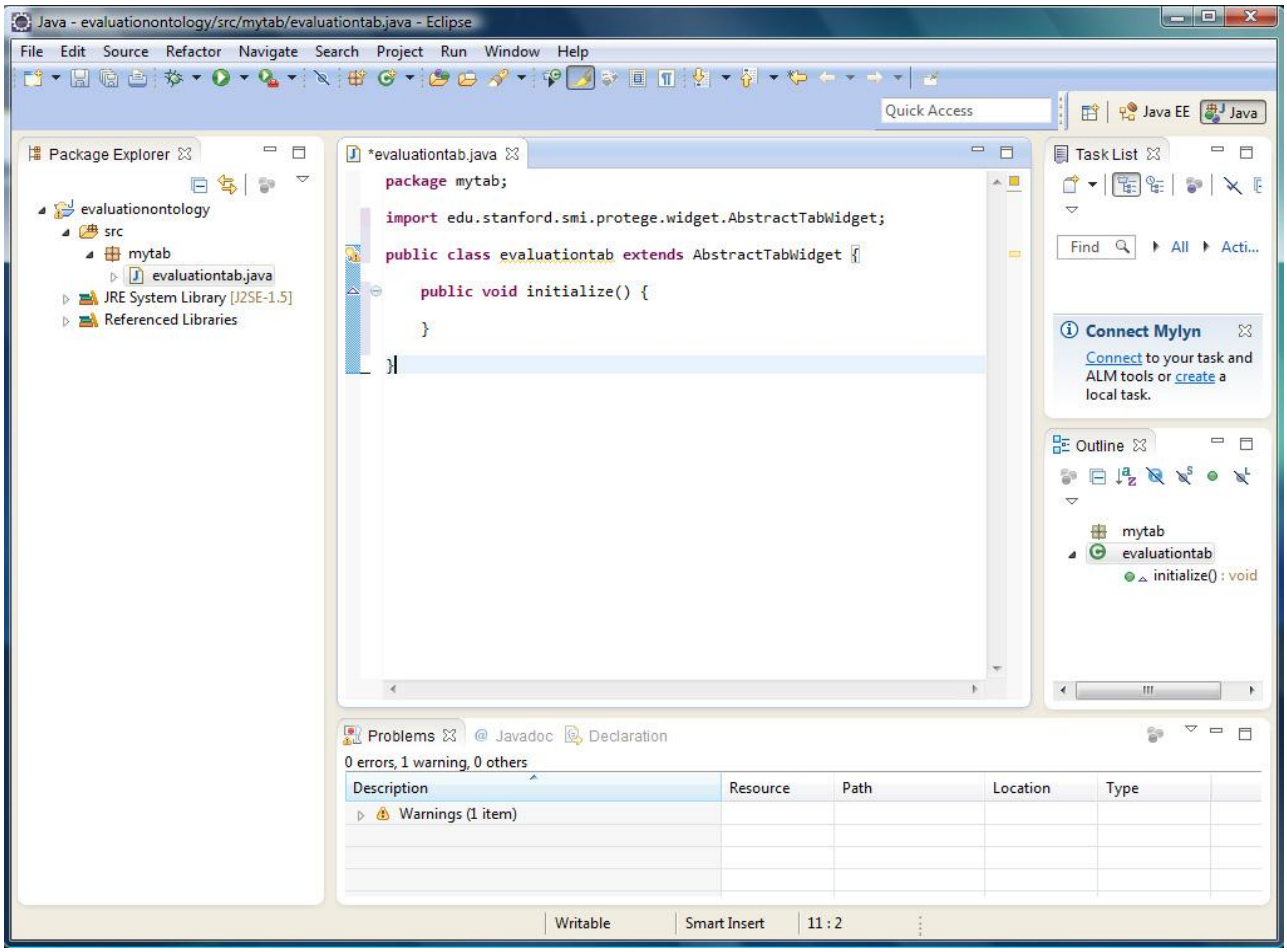


Picture 13: New Java Class window.

Click *Finish*. The new class should show up under the *src* folder. In order to create an empty implementation of the "initialize" method write the following code:

```
package mytab;

import edu.stanford.smi.protege.widget.AbstractTabWidget;

public class evaluationtab extends AbstractTabWidget {


        public void initialize() {

        }}
```

You should see this:



Picture 14: New created class.

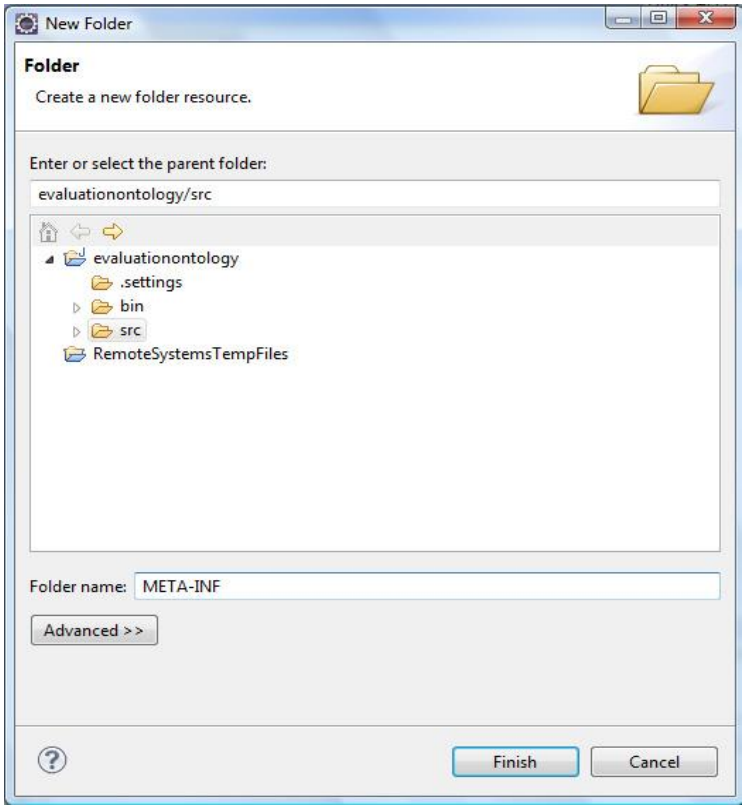
Step 4. Create the manifest file

To make Protege recognize the new tab widget, you will need to create a manifest file. There are instructions about creating the manifest file here (url: http://protegewiki.stanford.edu/wiki/PluginManifestsAndJars).

For our plug-in the code for the manifest file is:

```
Manifest-Version: 1.0

Name: mytab/evaluationtab.class
Tab-Widget: True
```

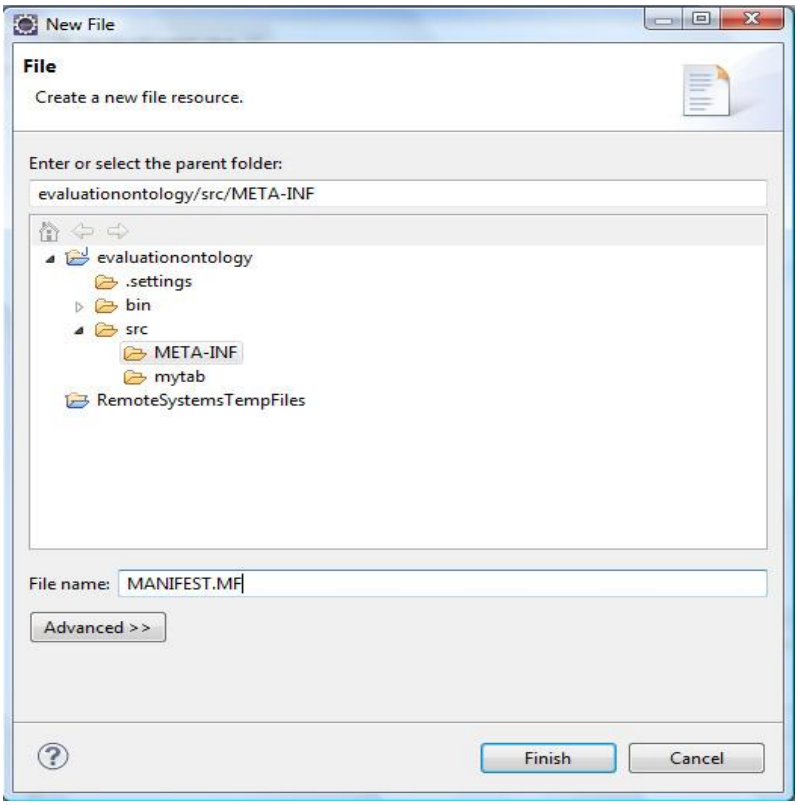
Right-click on src -> New -> Folder. In the panel, select the src node, and folder name: META-INF (capital letters!). This is what you should see:



Picture 15: New folder window.

After you click on *Finish*, the new folder *META-INF* is created under *src*.

Right-click on the *META-INF* folder, *New -> File*. In the panel, select *META-INF*, and for the file name write: *MANIFEST.MF* (capital letters!). This is how it should look like:



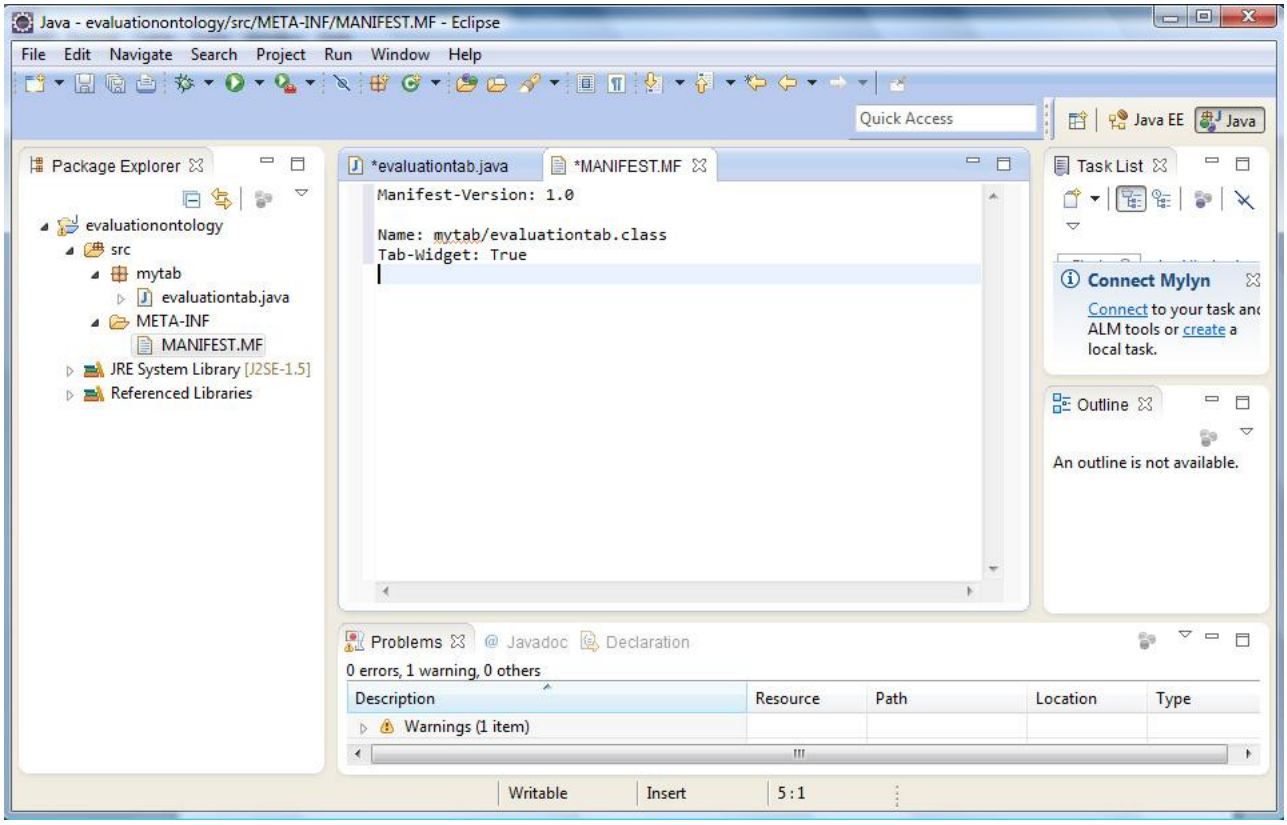
Picture 16: New file window.

Click *Finish*.

Copy and paste the text mentioned before in the content of the just created *MANIFEST.MF*

The empty row at the end of the file is very important!

Save the MANIFEST.MF file (by clicking Ctrl+S) and this is what you should see:



Picture 17: MANIFEST.MF code.

Step 5. Run the plug-in

So far, we have all we need to start the new plug-in. All we need now is to set up a run configuration for executing the plug-in.

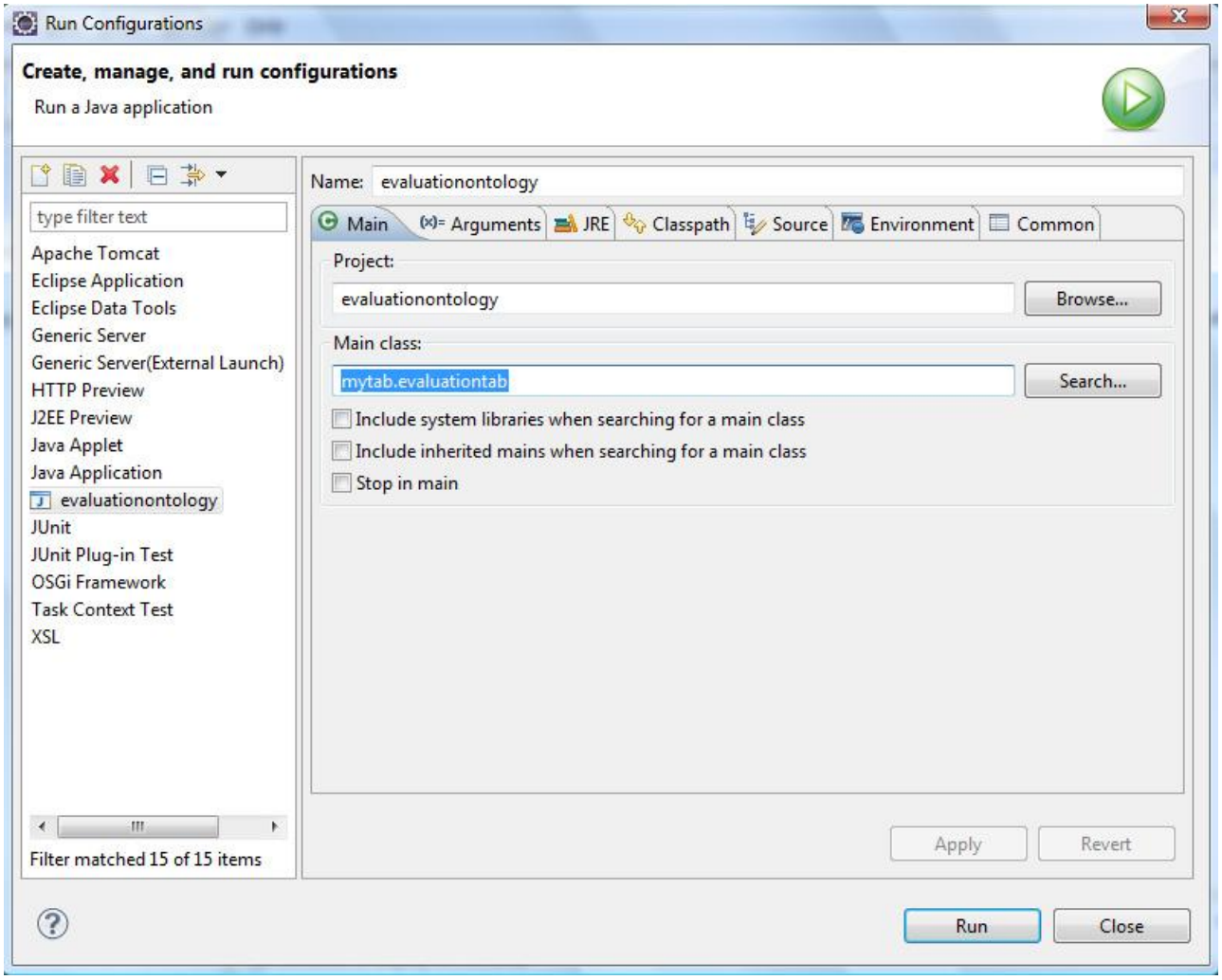
Go to Run Menu -> Run Configurations.... Create a new Java Application launcher, by selecting the Java Application in the tree and click on the create icon at the top of the panel. Fill in the following values:

Name: evaluationontology

Project: evaluationontology

Main class: mytab.evaluationtab

Eclipse will fill in automatically for you some of these values. This is what you should see:

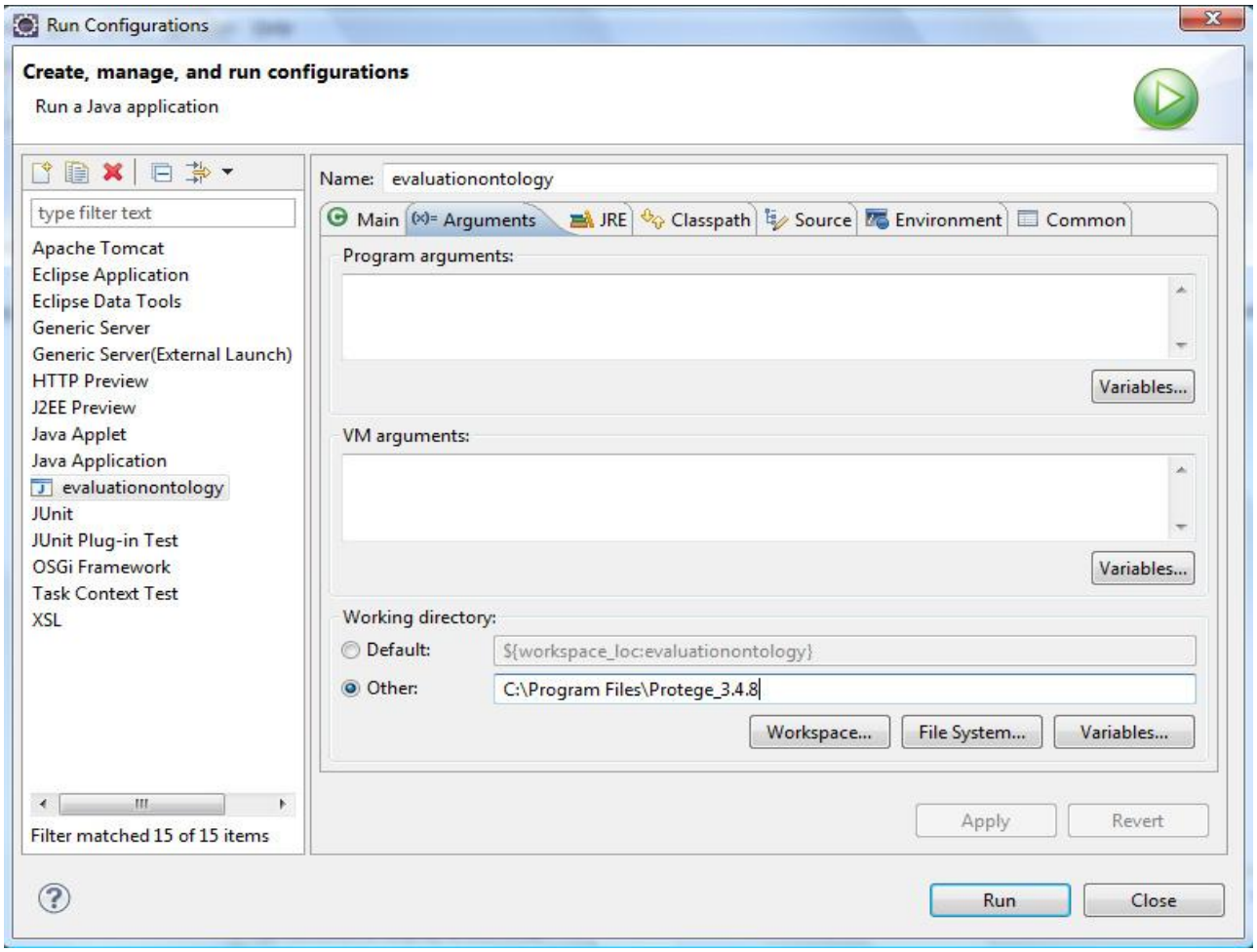


Picture 18: Run configurations window.

Then, switch to the Arguments tab, and for Working Directory select Other, then click on File System.. and select the path to your Protege installation directory . This is a very important step of the configuration.

In our example, Protege is installed in C:\Program Files\Protege_3.4.8 and this is the value that we

select. See screen shot from below:



Picture 19: Run configurations window.

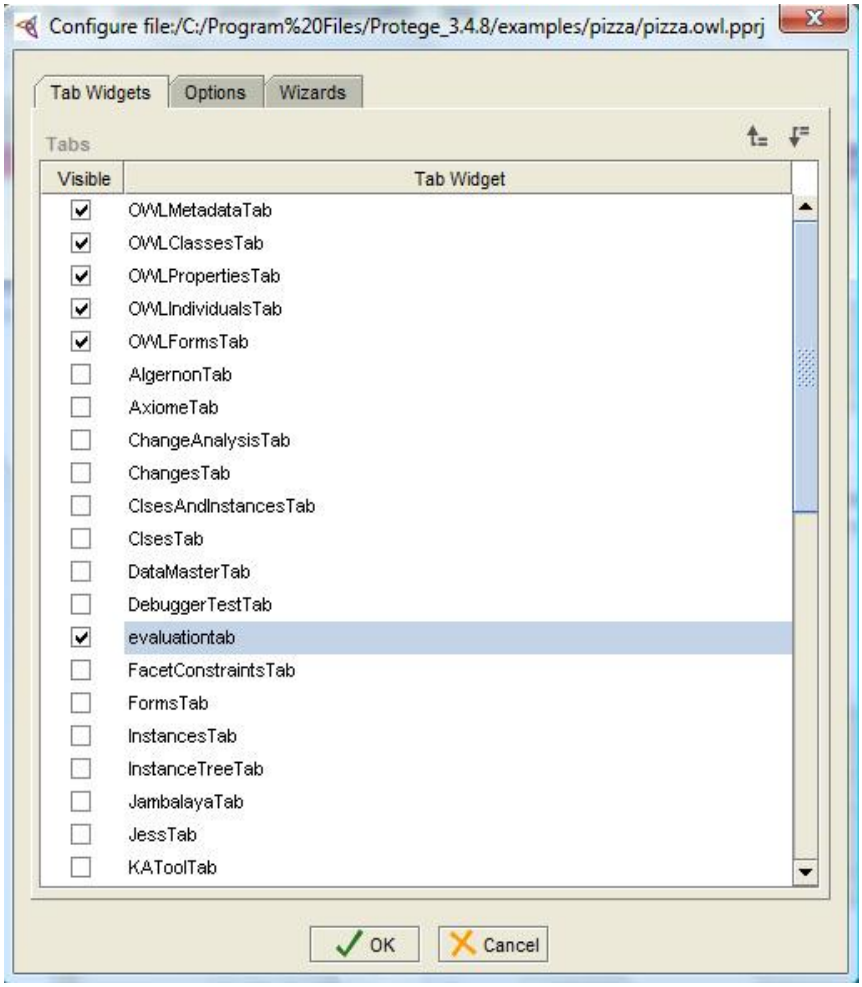
Click *Run*.

Once you click on Run, Protégé will start-up. You should see in the console view the messages about loading plug-ins. If you do not, it means you have misconfigured the Working Directory entry. Please check it again.

To test the plug-in, once Protégé has started, create a new project by clicking on the "New project" button in the Protégé welcome dialog, and in the next screen, select Protégé Files (pont and pins). You may also select any other backend, for example, OWL/RDF files.

Click on Finish.

A new project is created. To enable the tab that we have just created, go to Project menu -> Configure and in the Tab Widgets tab enable the “evaluationtab”. See below:



Picture 20: Selecting evaluationtab.

After you click OK, the new tab will show up in the Protégé UI. Congratulations! That's it!

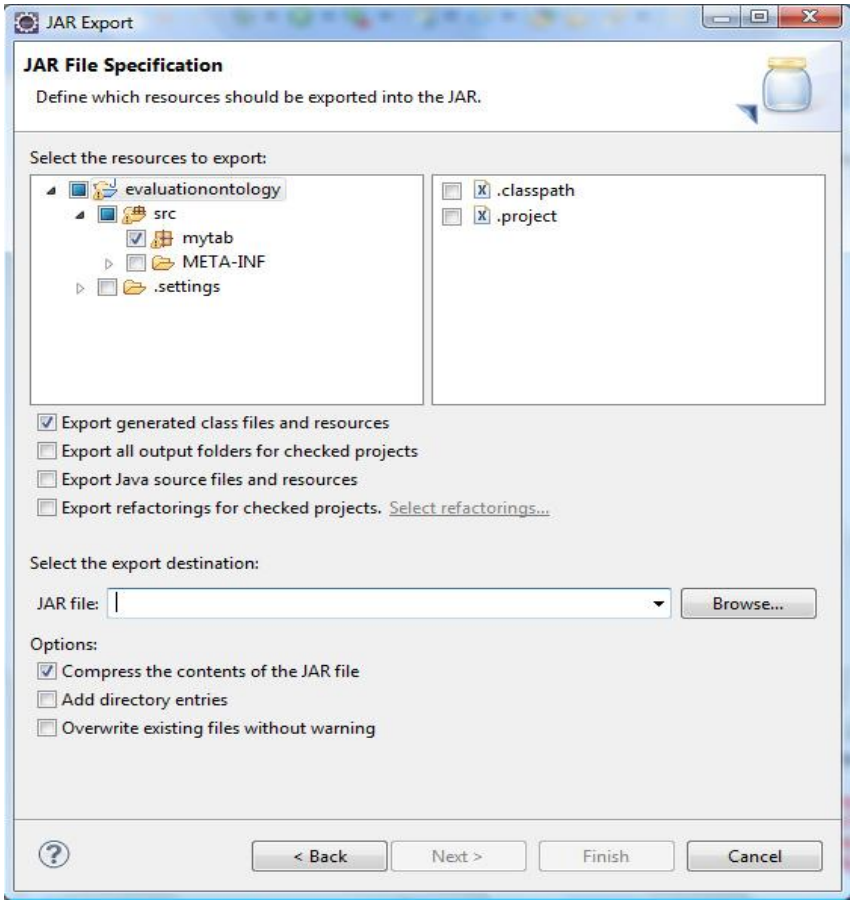
Creating a jar for your plug-in

If you're done with the plug-in development, or you would like to test the plug-in in a runtime Protégé, then you need to create a jar out of the plug-in classes and the manifest.mf (very important!) and copy the jar in a subfolder in the plugins folder of your Protégé installation.

Below are the steps for creating a jar.

Right click on src -> Export -> open Java -> JAR file -> click Next. In the next window, select src,

open the node and unselect META-INF (important!). Then add a path where the jar should be exported to. See below:



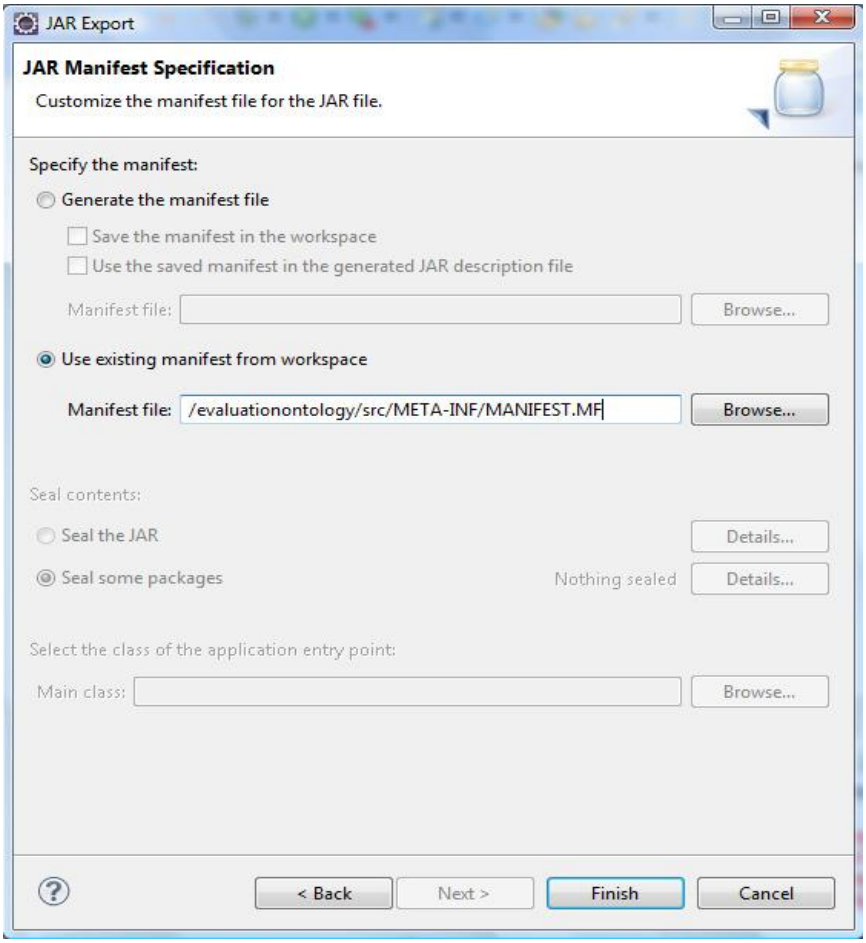
Picture 21: JAR Export window.

You could set as the path to which you export the plugin folder where the plug-in should actually be.

Click Next. In the next window, you can save the description of this jar in the workspace, in case you will want to generate later the jar again, by just one click (optional).

Then, click Next.

In the next window, you have to specify what manifest file should be used for the jar. Select Use manifest file from workspace and browse to the MANIFEST.MF in the workspace (In this example, /evaluationontology/src/META-INF/MANIFEST.MF). See below:



Picture 22: JAR Export window.

Click Finish. The jar will be generated in the location you have set up in the first export step.

To try out the plug-in in Protégé, you have to create a subfolder of plugins in your Protégé installation directory. For example, we could create a subfolder called: plugins/mytab. (Usually people use the Java package name as the subfolder name, but it is not required). Then copy the generated jar in this subfolder and start Protégé.

Note: If your plug-in depends on other plug-ins (e.g., protege-OWL plug-in, change management plug-in, etc.), then you need to create a plugin dependency file that you copy in the plugin subfolder

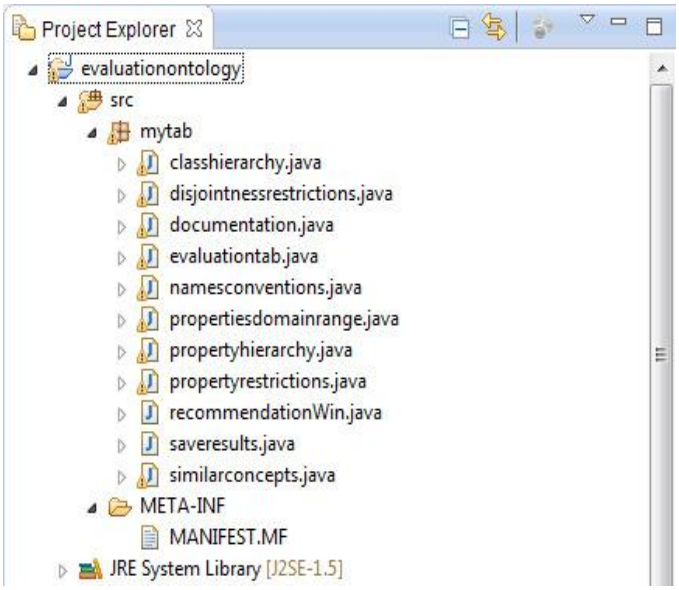
next to the plugin jar. Read details here (url: http://protegewiki.stanford.edu/wiki/PluginDependencies).

In our example the “plugin.properties” (which is a text file) has the following code inside it:

```
plugin.dependency.count=1
plugin.dependency.0=edu.stanford.smi.protegex.owl
```

Writing the code

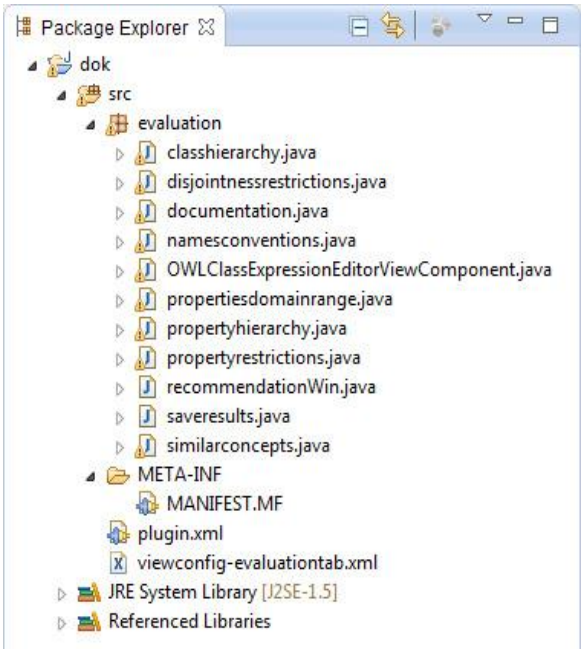
The next picture shows the java classes that implement the ontology evaluation plug-in.



Picture 23: Java Project structure.

Tab widget plug-in for Protégé 4.1

There are official instructions to set up the Eclipse so as to run the Protégé 4.1 from it on the following page(url:http://protegewiki.stanford.edu/wiki/CompileProtege4InEclipseFromSvn). After many trials we were unable to do the setup properly so we followed another approach in order to develop the plug-in for the Protégé 4.1. We just simply created a new java project on the Eclipse and after implementing the necessary files that are needed we exported the project to a .jar file just like we did previously for the 3.4.8 version of the Protégé. The structure of the project can be seen in the picture below:



Picture 24: Java Project structure.

As you can see the differences with the 3.4 version is the added files “plugin.xml” and “viewconfig-evaluationtab.xml”. These two files contains a declaration of the ways in which the plugin will extend the Protégé 4.1 capabilities and (more advanced) the ways in which the plugin capabilities can be extended by other plugins. The code for these files as well as the manifest file is written below:

MANIFEST.MF file

```
Manifest-Version: 1.0
Export-Package: evaluation
Bundle-ClassPath: .
Bundle-Category: protege
Bundle-Name: Evaluation
Bundle-RequiredExecutionEnvironment: J2SE-1.5
Require-Bundle: org.eclipse.equinox.registry,org.eclipse.equinox.commo
 n,org.protege.editor.core.application,org.protege.editor.owl,org.sema
 nticweb.owl.owlapi
Bundle-Version: 1
Bundle-ManifestVersion: 2
Bundle-Activator: org.protege.editor.core.plugin.DefaultPluginActivato
 r
Bundle-Description: A Plugin that makes ontology evaluation
Bundle-SymbolicName: evaluation;singleton:=true
Import-Package: org.osgi.framework,org.apache.log4j,javax.swing,javax.
 swing.border,javax.swing.colorchooser,javax.swing.event,javax.swing.f
 ilechooser,javax.swing.plaf,javax.swing.plaf.basic,javax.swing.plaf.m
 etal,javax.swing.plaf.multi,javax.swing.plaf.synth,javax.swing.table,
 javax.swing.text,javax.swing.text.html,javax.swing.text.html.parser,j
 avax.swing.text.rtf,javax.swing.tree,javax.swing.undo,org.w3c.dom,org
 .w3c.dom.bootstrap,org.w3c.dom.events,org.w3c.dom.ls,org.xml.sax,org.
 xml.sax.ext,org.xml.sax.helpers,javax.xml.parsers
Bundle-DocURL: http://www.co-ode.org
```

plugin.xml file

```
<?xml version="1.0" ?>

<plugin>

   <extension id="EvaluationTab"
           point="org.protege.editor.core.application.WorkspaceTab">
      <label value="Evaluation Tab"/>
      <class value="org.protege.editor.owl.ui.OWLWorkspaceViewsTab"/>
      <editorKitId value="OWLEditorKit"/>
      <index value="Q"/>
      <defaultViewConfigFileName value="viewconfig-evaluationtab.xml"/>
   </extension>

   <extension id="OWLClassExpressionEditorViewComponent"
                       point="org.protege.editor.core.application.ViewComponent">
      <label value="evaluationtab"/>
      <class value="evaluation.OWLClassExpressionEditorViewComponent"/>
      <headerColor value="@org.protege.classcolor"/>
   </extension>

</plugin>
```

viewconfig-evaluationtab.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<layout>
   <VSNode splits="0.3 0.7">

      <CNode>
         <Component label="Asserted hierarchy">
            <Property id="pluginId" value="org.protege.editor.owl.OWLAssertedClassHierarchy"/>
         </Component>
      </CNode>

      <CNode>
         <Component label="Evaluation Tab">
            <Property id="pluginId" value="evaluation.OWLClassExpressionEditorViewComponent"/>
         </Component>
      </CNode>

   </VSNode>

</layout>
```

If the reader want's to further study the anatomy of a Protégé 4.1 plug-in can check this page (URL: http://protegewiki.stanford.edu/wiki/PluginAnatomy).