

OnToology: An online tool for ontology documentation and evaluation

by

Ahmad Adel Alobaid

Submitted to the Department of Artificial Intelligence
in partial fulfillment of the requirements for the degree of

Master of Science in Artificial Intelligence

at the



UNIVERSIDAD POLITECNICA DE MADRID

July 2015

Supervisor

Oscar Corcho

OnToology: An online tool for ontology documentation and evaluation

by

Ahmad Adel Alobaid

Submitted to the Department of Artificial Intelligence
on July, 2015, in partial fulfillment of the
requirements for the degree of
Master of Science in Artificial Intelligence

Abstract

In this thesis, we designed and implemented an online tool (named "OnToology") to automatically generate documentation and perform evaluation for ontologies. It is also integrated with GitHub version control system to support collaborative environments.

Thesis Supervisor: Oscar Corcho

Acknowledgments

Firstly, I would like to thank my parents, grand parents, family and friends for the all prayers and support.

Special thanks again for my parents for everything over the years.

Thanks to the collaborators Daniel and Maria and to my supervisor Oscar for their guidance and support.

Contents

1	Introduction	15
1.1	Current Situation	16
1.2	Problem Statement	16
2	Background and State of the Art	19
2.1	Ontology Documentation	19
2.1.1	LODE	19
2.1.2	Widoco	21
2.1.3	Parrot	21
2.2	Ontology Visualization	22
2.2.1	VOWL2	22
2.2.2	AR2DTool	23
2.2.3	GrWOL	24
2.2.4	SOVA	25
2.3	Ontology Evaluation	25
2.3.1	OOPS!	26
2.3.2	MoKi	26
2.3.3	XD Analyzer	27
2.4	Online Ontology Development Tools	27

2.4.1	VoCol	27
2.4.2	WebProtégé	29
2.5	Version Control	29
2.5.1	GitHub	30
3	Research Questions, Assumptions, Objectives and Contributions	33
3.1	Question Addressed	33
3.2	Assumptions	34
3.3	Objectives	34
3.4	Contributions	34
4	Research Methodology	37
4.1	Design	37
4.2	Analysis	39
5	Implementation	41
5.1	Approach	41
5.1.1	Choosing Documentation Tool	42
5.1.2	Choosing Depicting Tool	42
5.1.3	Choosing Evaluation Tool	43
5.1.4	Suit Collaborative Environment	43
5.1.5	Integrating the Tools	43
5.2	Interaction with OnToology	45
5.2.1	Interface	45
5.2.2	GitHub	46
5.2.3	Configuration	46
5.3	Output Structure	46

5.4	Limitations	47
6	Evaluation	49
6.1	Experiment Steps	49
6.2	Results and Discussion	50
7	Conclusion and Future Work	51
7.1	Conclusion	51
7.2	Future Work	51
A	Tables	53
B	Figures	57

List of Figures

B-1	Questionnaire	59
B-2	Example of Repository Structure Before OnToology Triggering	59
B-3	Example of Repository Structure After OnToology Triggering	60
B-4	Request Example to OOPS! RESTFul Web Service	61
B-5	A Sample configuration file	62
B-6	OnToology page to register repositories	62
B-7	OnToology configuration example	63
B-8	Google trends comparison between Github vs Bitbucket vs GitLab[8] .	63
B-9	OnToology Architecture	64

List of Tables

A.1	VOWL2 Color Scheme	54
A.2	Evaluation Tools Comparison	54
A.3	Pitfalls Categoriza	55

Chapter 1

Introduction

Vocabularies as defined by the World Wide Web Consortium (W3C) are "the concepts and relationships (also referred to as 'terms') used to describe and represent an area of concern"[17]. Although there is no clear distinction between "ontology" and "vocabulary", W3C mention that, usually, "ontology" is used to refer to a more complex and formal collection of terms[17].

Ontology development is a complicated task. Therefore, researchers designed methodologies for ontology development. Usually, these methodologies adopt or recommend a set of activities or tasks to be followed. NeOn¹, Methontology² and most of the ontology methodologies, have documentation and evaluation activities[22].

Documentation generation and ontology evaluation is not simple and straightforward most of the time. Each is done separately and not in a collaborative manner. This requires a lot of effort, and becomes really exhaustive with large ontologies.

We develop an online tool named "OnToology" to automate the process of documentation generation, diagrams depiction and ontology evaluation. We also integrated

¹http://www.neon-project.org/nw/NeOn_Book

²<http://semanticweb.org/wiki/METHONTOLOGY>

the tool with a version control system to support collaboration.

In the remainder of this chapter, we talk about the current situation and the problems in it regarding ontology documentation and evaluation. In the next chapter, we talk about the state of the art about ontology documentation, visualization, evaluation and online ontology development tools. Then, we follow that with a background about version control systems. Next, we talk about the assumption we consider in this research followed by our objectives and our contributions to the research community. After that, we talk about the design and the analysis of the research methodology we adopted. Then, we present our implementation of OnToology. Finally, we show the evaluation of our work, conclude the research and propose future work.

1.1 Current Situation

Currently, ontology developers generate documentations, depicting diagrams and evaluate the ontology using separate tools for each. Even for tools that contain two of these three activities, its functionality is limited. Also the use of these tools is not done collaboratively in general. However, usually, ontologies are developed by a team rather than a single person. According to our research, we didn't find any tool for generating documentation, diagrams and evaluating the ontology in a collaborative environment. The only tool that is, in a way, close to what we are looking for, is VoCol, which is mentioned in the state of the art chapter later in this thesis.

1.2 Problem Statement

There are a couple of problems ontology development is facing today. We will focus on part of these problems that are more related to ontology documentation, diagrams

depicting and ontology evaluation in collaborative environment that we address below.

1. Ontology documentation, diagrams depicting and evaluation is done separately as existing tools are not integrated.
2. These tools are not designed to be used in a collaborative environment.
3. Each tool has a learning curve that developers need to invest time in.
4. Keeping track of the generated documentation, diagrams and evaluation for each version of the ontology is tedious for ontology developers.

Chapter 2

Background and State of the Art

2.1 Ontology Documentation

Ontology documentation is among the first activities to perform when looking at an ontology to understand and maybe reuse[44]. Writing ontology documentation manually require a lot of effort to write, maintain and reflect the updates and changes for newer versions of the ontology[44]. Hence, the need for tools to help generate ontology documentation automatically is needed. In this section, we talk about the most important documentation tools to us which are LODE[45], Widoco[28] and Parrot[12]. There are other tools that we don't cover them here like OWLDoc¹ and Neologism².

2.1.1 LODE³

LODE stands for Live Owl Documentation Environment. It is an online service to generate human-readable documentation for OWL ontologies in HTML format. It takes

¹<https://code.google.com/p/co-ode-owl-plugins/wiki/OWLDoc>

²<http://neologism.deri.ie/>

³All the information in this section is taken from [45]

into account the ontological axioms and annotations following the W3C recommendation for the appearance and functionality.

LODE extracts classes, axioms, properties of data, objects and annotations from any OWL file. It also extracts meta-modelling, SWRL [32] rules and namespace declarations. Then, it will produce a single human-readable HTML page with embedded links for browsing and navigation.

The document starts with a title extracted from the OWL file. Then, it lists the metadata of the ontology like the URI, authors, publication date and imported ontologies. Next, an abstract, table of content and introduction section is followed. After that, it renders all classes, objects, data, name individuals, general axioms, SWRL rules and namespaces.

For imported ontologies, LODE renders all annotations and axioms defined in the imported ontology if the parameter "closure" or "imported" is provided. For the axioms inference, LODE relies on Pellet [48]. So when the parameter "reasoner" is specified, Pellet will infer axioms from the ontology and LODE will render them in the documentation page. In this page, within each entity definition, axioms and restrictions are rendered using Manchester Syntax [31]. URIs built with LODE are compliant with "Cool URIs"[23]. This allows LODE, beside the usage of content-negotiation, to automatically represent ontologies and their entities in the HTML page. Since the annotation property allows referring to images, LODE render these images as part of the HTML documentation page. LODE also supports other languages besides English (which is the default) by specifying the "lang" parameter. Which in-turns, will be used in the documentation page as the default language, assuming appropriate language annotation are available in the ontology.

2.1.2 Widoco⁴

Widoco stands for WIZard for DOcumenting Ontologies. It uses LODE to extract information about the ontology such as classes, properties, annotations, axioms and namespaces. It generates the documentation into separate sections in separate HTML files (different than LODE which generates the documentation into a single HTML file). It automatically annotates the documentation with RDF-a⁵ after it does the extraction of meta-data from the ontology. It also provides guidelines on how to fill the main sections of the documentation. In addition, Widoco generate the documentation associated with a provenance page⁶ which (according to W3C) is "information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness"[30].

2.1.3 Parrot⁷

Parrot is a RIF(Rule Interchange Format)[35] and RDF ontology documentation tool. It is implemented in Java and is based on Jena⁸, Java-RDFA⁹ and RIFLe¹⁰ which parses annotations in RIF document and express them as RDF graph. Parrot is available as Eclipse plugin¹¹, NeOn Toolkit plugin¹², website that is accessible through a web browser and as a REST web-service[13]

Parrot generates table of contents, summaries and glossaries and include it in the generated documentation. It also depicts RIF in a detailed view showing RIF rules. In

⁴All information related to Widoco is taken from [28]

⁵<http://www.w3.org/TR/rdfa-syntax/>

⁶<http://www.w3.org/TR/prov-o/>

⁷Information about Parrot are taken from [50], [12] and [14]

⁸<http://jena.apache.org/>

⁹<https://github.com/shellac/java-rdfa>

¹⁰<https://bitbucket.org/fundacionctic/rifle/wiki/Home>

¹¹<https://eclipse.org/>

¹²<http://neon-toolkit.org/>

addition, it deals with redundant and conflicting annotations. It also handles multi-lingual annotation and gives the user the option to choose the language of interest for the documentation.

2.2 Ontology Visualization

Ontology visualization is an important activity for understanding an ontology and hence, ease the development and reusing of ontologies. According to [39]. Ontology depicting tools like OWLViz¹³, OntoTrack [38], OWL-VisMod [41], Jambalaya [49] and KC-Viz [40] only show class hierarchy while OWLViz, OntoGraf¹⁴ and FlexViz [26] don't show datatype properties. In [39], the authors also mention that TGViz [20] and NavigOWL [33] has links and nodes that looks alike expect for the color not like GrOWL [36] and SOVA[37] that uses different shapes, symbols and colors. For the reasons mentioned above, we will talk about the best candidates in our opinion, which are VOWL2, AR2DTool[43], GrOWL and SOVA.

2.2.1 VOWL2

VOWL2 is a visual notation for OWL ontologies. It targets ontology experts as well as users new to ontologies.

Ontologies are represented as graphs in VOWL2. It assigns graphical primitives to ontology elements i.e. classes as circles and relations as labeled arrows. As the visualization of instances is not extremely important in most cases, details about individuals are not included in VOWL2. An exception to that, is number of individuals of classes, which is proportional to circles diameters (larger diameter for larger number of

¹³ <http://protegewiki.stanford.edu/wiki/OWLViz>

¹⁴ <http://protegewiki.stanford.edu/wiki/OntoGraf>

individuals). To further improve the visualization, VOWL2 use default color scheme to help distinguish different elements (see figure A.1).

The user interface is composed of three parts, a *viewer* to display the visualization, a *sidebar* to list details about the selected element and the *controls* to export the visualization and adopt the graph layout.

VOWL2 has been implemented in two tools, ProtégéVOWL and WebVOWL. ProtégéVOWL is a java plugin for Protégé¹⁵. It is based on Prefuse[34], a visualization toolkit. It uses Protégé’s data model. WebVOWL is a standalone web-based application based on D3[25] visualization library. It also uses open web standards, so it doesn’t need proprietary plugins. WebVOWL allows exporting the diagram (or part of it) as an image in SVG¹⁶ format. It support touch interactions to do things like zooming, using two fingers zooming gestures. WebVOWL also made sure that even that there are feature that are not available on touch screen, it won’t be crucial for the visualization.

In both implementations, it generate force-directed graph layout (to reduce cross over links). It allow users to zoom in to a certain parts for details or zoom out to see the global structure of the ontology. It also supports highlighting and displays extra information once an element is selected.

2.2.2 AR2DTool

AR2DTool[43] is an ontology visualization tool to draw diagrams based on Jena API¹⁷ for parsing ontology files and GraphViz¹⁸ for drawing diagrams. The tool has a configuration file that allow customizing the style of the output like the shape of classes,

¹⁵<http://protege.stanford.edu/>

¹⁶https://en.wikipedia.org/wiki/Scalable_Vector_Graphics

¹⁷<http://jena.apache.org/>

¹⁸<https://github.com/jabbalaci/graphviz-java-api>

color of classes shape, the size of the image and whether to include the full URI or show the prefix instead. It also has the facility to choose what elements to include in the generated file and what elements to ignore. The format of the diagram file has to be provided as a command line option e.g. "png" or "pdf" as well as the configuration file to be used by AR2DTool.

2.2.3 GrOWL¹⁹

GrOWL is based on the Prefuse[34] visualization library. GrOWL is implemented as a Java applet²⁰ plugin to Protégé and as a standalone Java application. The former implementation uses Jena API while the latter uses a trimmed version of WonderWeb OWLAPI. GrOWL has an ABox view, a TBox view and an RBox view. ABox contains role assertion between individuals, TBox about concepts and RBox about roles and roles hierarchy [27].

GrOWL supports automatic and manual layout for elements positioning. For automatic layout, GrOWL uses animated force directed layout that allows auto-position the elements in an interactive way with the animation. It also has manual layout to allow the user to manually position the elements.

GrOWL has a tree based navigation. It has the list of classed and instances in a class tree (similar to Protégé). It allows the user to click on any object in the class tree, then will adjust the view by centering it on the selected element. Although GrOWL provided a good results, it does not guarantee clarity for large ontologies.

¹⁹All the information in this section is taken from [36]

²⁰<https://docs.oracle.com/javase/tutorial/deployment/applet/>

2.2.4 SOVA²¹

SOVA stands for Simple Ontology Visualization API. It is developed as a Protégé plugin for ontology visualization. It offers two types of visualization, full ontology visualization, and hierarchy of classes and individuals visualization.

In the full ontology visualization, it presents all classes, individuals, relations and properties. On the side panel of SOVA, it allows filtration of elements of the ontology. It has distance slider to allow hiding ontologies far from a selected element. It also has the option to choose the arrangement algorithm. It supports two arrangement algorithms, ForceDirect algorithm (which is the default algorithm) and RadialTree algorithm. ForceDirected algorithm use gravity algorithms while the RadialTree algorithm places ontology elements as circles around the selected element.

For the classified hierarchy of classes and individuals visualization, user have to choose "Hierarchy Tree" and the reasoner from SOVA tab, then click on "start" to start the reasoner. Next, the user has to press the "restart" button to see the visualization.

2.3 Ontology Evaluation

"Just as testing is an integral part of software engineering, so is ontology evaluation an integral part of ontology engineering" [42]. As the quality of ontology can be affected by the existence anomalies in the ontology[47], identification of anomalies or bad practices in ontologies is one of the crucial issues in ontology evaluation. In this section, we will talk about OOPS![47], MoKi[42] and XD Analyzer[24]. There are also other evaluation tools like OntoCheck²² and Jena Eyeball²³, but we mention the ones with more pitfalls covered according to[47] (see A.2 and A.3)

²¹All the information in this section is taken from [37]

²²<http://protegewiki.stanford.edu/wiki/OntoCheck>

²³<https://jena.apache.org/documentation/tools/eyeball-getting-started.html>

2.3.1 OOPS!²⁴

OOPS! (OntOlogy Pitfall Scanner) is a web-based tool for detecting pitfalls, based on Jena APIs.

It consists of a single page that can be accessed through any of the major web browsers without the need of any extra software or plugins to be installed. The user can pass the ontology to OOPS! by specifying the URI of the ontology or by copying the content of the ontology file to the text area available in the page. Next, it will detect errors in the ontology, add warnings about RDF syntax and provide modelling suggestions. Then, it will list the errors, warning and suggestions with a brief description of each, number of occurrences and affected elements. OOPS! has a catalogue of pitfalls it can detect [11].

2.3.2 MoKi²⁵

MoKi (MOdeling wiKI) is a collaborative ontology modeling tool based on Media-wiki²⁶. It adopts ontology evaluation to automatically check an ontology for compliance with modeling guidelines to detect potential modeling errors. The evaluation of the ontology happens iteratively, while the ontology being developed. MoKi has a checklist of modeling guidelines where it checks ontology elements against and view how much does ontology elements comply with the modeling guidelines.

²⁴All information about OOPS! are taken from [47]

²⁵Information in this section are extracted from [42] and [29]

²⁶<http://www.mediawiki.org>

2.3.3 XD Analyzer²⁷

XD Analyzer is one component of XD (eXtreme Design) Tools, which is a plugin for NeOn toolkit²⁸ composed of components that support pattern-based design. Ontology pattern design is "a modeling solution to solve a recurrent ontology design problem" [10]. XD Analyzer provides feedback to the user with respect to the best practices of ontology design. The feedback can contain errors (i.e. missing type), warnings about the bad practices (i.e. missing labels) and suggestions for improvements (i.e. suggest inverse property for objects with no inverse property).

2.4 Online Ontology Development Tools

In this section, we talk about two online ontology development tools, VoCol and WebProtégé. Which are more related to the scope of this research than other tools like Neologism²⁹.

2.4.1 VoCol

VoCol[46] is a tool and a methodology inspired by agile development. A number of companies, domain experts and developers are involved in MobiVoc³⁰, which is the project that triggered VoCol creation and hence, the requirements of VoCol is tailored accordingly. The requirements include validation of ontologies (syntax and semantics), document generation and publication of ontologies as linked open data (LOD). Versioning control is needed to keep track of the decision taken beside older versions. It is also required for the environment to be accessible through web browsers

²⁷Information in this section are from [24] and [10]

²⁸<http://neon-toolkit.org/>

²⁹<http://neologism.deri.ie/>

³⁰<http://www.mobivoc.org/>

(because of the non-technical users involved), as well as rich clients, which offer more advanced features.

VoCol features include:

- Easy quick editing: anyone can edit the ontology without restriction, which makes it easier to have errors and easier to fix them as well.
- Works on browser: without the need of another tools or plugins to be installed
- Versioning control: for tracking of changes of the vocabularies and handling editing conflicts.
- Automatic publication: for any valid vocabulary.

VoCol use Git as the underlying versioning control engine with GitHub and GitLab as front-ends. GitHub allow them to host the files on it, while GitLab allows them to self-host the files. These front-ends allows users without enough knowledge to use Git, to contribute using web browser to edit the files for authoring.

After that, PyGitHub [15] is used to interact with GitHub in regular time intervals and check for new commits. Then, it validates files affected by the last commits and notifies users involved in these commits relying on the issue tracker of GitHub to sort out the responsibilities of the issues if any. The issue has an error message as the issue title and in the body of the issue it contains a link to the file containing the error with highlighting of the line having the error. The issue body also includes links of commits and the contributors of the commits in this validation round.

After passing the validation process, VoCol produce two versions of the vocabulary, machine friendly version and human friendly version. Before the generation, it merge Turtle files into a single file. Then, it produce from that file the vocabulary in RDF/XML as machine friendly version. For the human friendly version, it produces

HTML+RDFa from that single file using schemaorg as documentation generator. Using a local server to serve both versions of the documentation and made the vocabularies available under their URIs with content negotiation between the HTML and RDF/XML using some configuration on the web server. For visualization, VoCol uses WebVOWL, which is an implementation of the Visual Notation for OWL ontologies (VOWL), to provide graphical depictions of the OWL elements.

2.4.2 WebProtégé

WebProtégé is a light weight ontology editor and knowledge acquisition with online collaboration functionality [51][46]. It is an online version of Protégé³¹ ontology editor, with a subset of its features[46]. It targets ontology developers and domain experts. It allows the users to customize the interface including the layout to suite their needs and purpose as developers or domain experts. It also offers knowledge acquisition using special kinds of forms which allow the association with form fields (e.g. text fields, checkboxes, etc.). Besides these features and the ability to be extended via plugins, the development of WebProtégé is still in progress and they advice not to develop plugins against yet [46][18].

2.5 Version Control

Version Control is a system to record changes on files and keep a history of the files overtime so any version of the files can be retrieved later [1]. Most certainly people working on projects need it as they want to keep track of their files so if they missed up of lost files they can revert to an older version of the project[1].

³¹<http://protege.stanford.edu/>

At some point people need to collaborate with other team members, so the idea of centralized version control systems were developed i.e. CVS (Concurrent Versions System)³², Subversion³³, and Perforce which now supports even distributed version control³⁴[1]. The problem with this approach is that it has a single point of failure and if no local copy is kept, everything will be lost if the center server's hard drive becomes faulty[1].

Distributed version control systems on the other hand(i.e. Git[3], Mercurial³⁵, Bazaar³⁶ or Darcs³⁷), keeps a full backup of the project and all the history locally, so if the server's hard drive is got corrupted, the project can be fully restored once the server is up again (or another server is used)[1].

2.5.1 GitHub

Git is a free and open source distributed version control system[3]. GitHub³⁸ is "a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features"[5].

A repository, is where a project is stored and can be thought of as the project's folder[6]. In GitHub, a repository can belong to a GitHub user or an organization. Organizations are owned by one or more GitHub users. GitHub allows users to work on projects collaboratively by allowing the owner to add collaborators to the repository[7].

A repository has one or more branches. A branch, is a new line that diverge from

³²<http://www.nongnu.org/cvs/>

³³<https://subversion.apache.org/>

³⁴<http://www.perforce.com/>

³⁵<https://mercurial.selenic.com/>

³⁶<http://bazaar.canonical.com/en/>

³⁷<http://darcs.net/>

³⁸<https://github.com/>

the main development line of the repository, and changes on this new line are not reflected on the main development line[4]. To save changes of a project belongs to a branch, "commit" command must be used[6]. For the changes to be reflected on a repository hosted on GitHub, the user must use "push" command[6].

Users can propose changes for project files and its content in GitHub by creating a "pull request". To do so, the user must "fork" the repository first. Forking a repository will create a copy of the repository and will have this copy placed under his user's account[6]. Then, the user can push changes to the forked version of the repository he has under his account and create a pull request. After that, collaborators of the repository can see the proposed changes, if they choose to accept the proposed changes, they should "merge" the repository. The merge can be done from GitHub website if there are no conflicts, otherwise merge can be done locally on the user's machine and resolve the conflicts before the merge.

Chapter 3

Research Questions, Assumptions, Objectives and Contributions

In this chapter, we address research questions, then we mention the assumption we will agree upon in this research. Next, we talk about the objectives. After that, we present our contribution.

3.1 Question Addressed

- Can manual work (or part of it) be eliminated/automated in ontology development process today
- Is it efficient to automate part of the manual work done in ontology development.
- Can the automation of some activities in ontology development be achieved in collaborative environment
- Does the automation of producing documentation, diagrams and evaluation reports makes the work on ontology developer easier

3.2 Assumptions

- The ontology is written in OWL or RDF/S with RDF/XML or Turtle serialization.
- The ontology is written in a correct format. Ontologies contain errors won't be parsed correctly and therefore, won't produce the correct output.
- The ontology is hosted on GitHub and the user already has access to the ontology in GitHub.
- In case the application is hosted by the user, we assume the operating system used is supported by the libraries we are using.

3.3 Objectives

- Ease the development of ontologies in collaborative environment.
- Increase the productivity of ontology development and develop ontologies in less time.
- Help develop ontologies with fewer syntax and semantic errors.

3.4 Contributions

- C1: In collaborative environment, we ease the development of ontologies by automatically invoking tools for generating documentations, diagrams and evaluation report every time the developers save the ontology to the server so developers don't have to call these tools every time they made a change to the ontology.

- C2: Increase the productivity of ontology development by automating part of the tasks developers usually do manually like generating documentation, diagrams and evaluation report, which allow them to finish the development of the ontologies faster.
- C3: Help develop ontologies with fewer errors by automatically generating evaluation reports for every GitHub push (save point) of the ontology which shows pitfalls while the ontology is being developed, and hence, helps discovering them in early stage of the development.

Chapter 4

Research Methodology

In this chapter we talk about how we approach our research and the research methodology we adopted in our study, which is qualitative research methodology. We used a questionnaire that was answered by researchers from the Ontology Engineering Group at Universidad Politécnica de Madrid, which is a respected research group in the research community and has a large experience level of developers and researchers. After that we analyze the survey results and validate our approach.

4.1 Design

We notice that ontology developers generate documentation for every version of their ontology, they also generate diagrams and evaluation report. This manual process is not as easy and simple as it sounds, it is consuming developers time to write and execute the command and choose the configuration for generating documentation and diagrams. They also has to keep track of the documentation, diagrams, evaluation report and whether they belong to the latest version of the ontology or they should generate it for the latest version of the ontology. Also these steps have to be repeated

for every newer version of the ontology. Add to that, human errors include choosing the wrong configuration or the wrong ontology file version. So, we decided to overcome this problems by automating the generation of documentation, diagrams and evaluation report in a tool that we called "OnToology". This approach will not solve all the problems we addressed in this section. We started to look deeply into the problems and how ontology developers and researchers work, we notice that they use version control system and hosting called "GitHub". We decided to integrate the tool with the version control system so, every time a new version of the ontology is generated or a change is made to the ontology, the tool will be triggered and generate documentation, diagrams and evaluation report. With this, all the problems addressed in this section will be solved.

Now we need to validate the approach we decided to take to tackle the problems we mentioned earlier. We adopt qualitative methodology as our research methodology and we decide to use survey to gather the information we need for our study. Then, we choose who we are targeting in this survey. Our target are researchers and developers of ontologies. We choose researchers and developers from the Ontology Engineering Group at Universidad Politécnica de Madrid. It is a well known research group in the research area for its contribution, so the answers given by the researchers from the group can be considered as trustworthy. This research group covers a wide range of researchers and developers from beginners to professionals, this type of diversity allows us to cover a larger span of expertise in our study. The reasons we mentioned for targeting Ontology Engineering Group adds strength and soundness to our methodology. Next, we design our questions (see figure B-1) to be open-questions and needs around five minutes to be completed. We choose the questions to be open-questions to give researchers/developers the freedom to express their opinion as we do not want to limit their answers, as the respondent may mention something we already missed [16][19].

4.2 Analysis

The overall results we got from the survey was as we expected before the survey. All participants agreed that automating some ontology engineering activities is helpful and some participants commented that "It is quite obvious". One participant emphasised that even with having some parts of ontology activity automated, human intervention is quite crucial. Another one said "[it is] extremely helpful when a set of ontologies for Linked Data development have to be maintained over time". So, its importance increase as the ontology is being developed.

Automating the generation of documentation and diagrams and evaluating the ontology, received a full agreement from the participants as well. Some said it would save time. Another participant said that, they would need to write scripts otherwise. We can see here, that participants believe strongly on automating some parts of the ontology development activities, namely the generation of documentation, diagrams and ontology evaluation report.

Having the automation tool (that generates documentation, diagrams and evaluation report) in a collaborative environment is helpful according to most of the responses. A response we got mention that, having the tool in collaborative environment is helpful, because most ontologies are created by multiple users. Another response that we got, the participant mentioned that, integrating the automation tool with GitHub will ease the ontology development. All participants also agree that, automating part of the ontology development activities will increase the productivity as well. Another response we got agrees that it would increase the productivity if "it is done well".

While a single participant was not familiar with the evaluation report, the rest of the participants agreed that generating the evaluation report during ontology development stage will help detect pitfalls of the ontology in an earlier stage and will help producing

ontologies with fewer pitfalls.

For the generated documentation, diagrams and evaluation report, most participants prefer to have them in a pull request rather than a direct push. They said, because they want to see the changes, review them and see whether to accept them or not. One participant, who prefers direct push commented that, the reason for him to prefer the direct push is, to have the documentations in the repository always up to date. We argue that, ontology should be correct before having the documentation and diagrams available, otherwise the up-to-date documentation and diagrams will include the errors of the ontology (if any), not to mention that some errors will result in generated files to be empty. So, obviously, not the way to go.

To have the control to enable/disable certain activities, namely, generating documentation, generating diagrams and/or evaluating the ontology through a configuration file, got the agreement from all participants. One participant said that he prefer to have the configuration file if it is clear. All participants also agreed on having the configuration in the webpage beside having the configuration file. Most of them said that they agree because it would be easier for the users. We can conclude from participant responses to this regard that having the automation tool easy to use is crucial to all users even ontology developers with technical background.

According to the answers we received from the participants, It met our overall expectations and we will be implementing the system accordingly, taking into account the participants concerns and suggestions.

Chapter 5

Implementation

OnToology is developed using Python¹ programming language and Django² which is a Python Web framework³. We also use MongoDB⁴ as the database engine.

In this chapter, we talk about the approach we followed in implementing OnToology. We also explain the usage of OnToology, the configuration, the output structure and the limitation of the tool.

5.1 Approach

In this section, we choose the tools for generating ontology documentation, diagrams and ontology evaluation. We also mention the reasons behind choosing Widoco, AR2DTool and OOPS! beside the good support we received from the developers of the tools. Then, we talk about integrating the tools and then how did we manage to suit them to collaborative environments.

¹<http://www.python.org>

²<http://www.djangoproject.com>

³https://en.wikipedia.org/wiki/Web_application_framework

⁴<http://www.mongodb.org/>

5.1.1 Choosing Documentation Tool

For the documentation tool we choose Widoco. We choose Widoco for the following reasons. Widoco is based on LODE, which is widely used and known in the literature. LODE uses a single HTML file with all the documentation inside, but widoco separate the documentation into sections, each in a separate HTML file (but all sections are viewed in single page). Having separate HTML file for each section ease the editing of the documentation, specially with large documentation. It automatically annotates the generated documentation with RDF-a. It also produces provenance page. Integration with Widoco is seamless, as it can be executed through command line interface(CLI) with parameters and no further interaction is needed

5.1.2 Choosing Depicting Tool

For ontology depicting, there are a lot of tools available for that. This make it harder for us to choose and evaluate each of these tools. Luckily for us, VOWL2 did a very nice comparison between ontology visualization tools. Since VOWL2 was the winner of this comparison, obviously, we will compare it to a AR2DTool which is developed in Ontology Engineering Group⁵ in Universidad Politécnica de Madrid (see 2.2.1 for more information about VOWL2). Although WebVOWL (which is an implementation of VOWL2) is rich in terms of features, it does not convenient for us to integrate with. The reason is that, WebVOWL needs the user to convert the ontology to VOWL-JSON format before using WebVOWL. It also has a graphical interface which require user interaction, which makes it hard to integrate with. AR2DTool on the other hand, can be called through CLI with parameters without the need of human interaction. It has a configuration file to allow changing the properties of the diagram i.e., colors,

⁵<http://www.oeg-upm.net/>

shapes and ignore elements with given URIs. These are the main reasons for choosing AR2DTool for ontology visualization.

5.1.3 Choosing Evaluation Tool

We choose OOPS! to integrate with for ontology evaluation. It is used in a lot of projects and got a lot of positive feedback. It is also online based with REST web-service provided, so no installation is needed and easy to integrate with. Furthermore, it detects the largest number of pitfalls (see 2.3.1) and outperforms both MoKi and XD Analyzer (see section 2.3 for more information about ontology evaluation tools).

5.1.4 Suit Collaborative Environment

To do so, we choose to integrate with GitHub (see section 2.5 for information about GitHub and version control systems). GitHub by definition, is for people to work collaboratively on projects. So, we inherit this feature by integrating with GitHub. Since, GitHub offers different kinds of collaborations, we inherit all kinds of collaboration options offered by GitHub. There are other version control systems like Bitbucket and GitLab, but GitHub has the highest interest according to google trends (figure B-8).

5.1.5 Integrating the Tools

After choosing the tools, we need to integrate the tools in a way. To do so, we develop a software to integrate with the other tools. We named the software "OnToology" and design it to acts as "command and control center" as it is the one responsible for invoking and interacting with other tools.

For integrating OnToology with Widoco and AR2DTool, we use command line

interface (CLI) through *subprocess*⁶ Python library. Both, Widoco and AR2DTool are Java⁷ based and available in JAR format. JAR applications require Java Runtime Environment (JRE)⁸ to be executed, so we install JRE on our server.

Calling Widoco for the first time requires an extra argument to be passed, that inform Widoco there is no configuration file, parse the ontology and generate the configuration file. We also pass the ontology and the configuration file if it does exist.

For calling AR2DTool, OnToology provide a default configuration file for each diagram type if it doesn't exists. We generate two kinds of diagrams by calling AR2DTool twice, each time with a different configuration file.

OnToology is integrated with OOPS directly and indirectly. Directly, through OOPS! RESTful Web Service to generate a summary of ontology evaluation, which will be included in the GitHub issue that will be created once a change in the ontology is pushed to GitHub. The request is sent as an HTTP POST⁹ with the ontology file. A request example for a sample ontology¹⁰ is in figure B-4. Indirect integration with OOPS! is through Widoco, by passing an extra argument "-oops" to generate the evaluation report.

We integrate OnToology with GitHub using PyGithub Python library. The reason we are using this specific library, is because it is one of the Python libraries listed in GitHub official website that implements most of GitHub APIs. Although PyGithub provides a quite range of APIs, it is still not sufficient for our needs. Therefore, OnToology uses Git application¹¹ as well through CLI. The architecture of OnToology and the integration with the tools are shown in figure (B-9).

⁶<https://docs.python.org/2/library/subprocess.html>

⁷<http://java.com>

⁸<http://www.oracle.com/technetwork/java/javase/overview/index.html>

⁹<http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>

¹⁰<http://www.linkdatatools.com/introducing-rdfs-owl>

¹¹<http://git-scm.com/>

There is also a set of steps that need to be done once. First, we create a GitHub user (we name it "OnToologyUser"). Second, we need to create a GitHub application[9] (we name it "OnToologyApp"). Next, we generate ssh key from the server and add it to OnToologyUser as mentioned in GitHub[2].

5.2 Interaction with OnToology

In this section, we talk about the interaction with OnToology. There are three ways users interact with OnToology, through the interface, Github pushes, and configuration files.

5.2.1 Interface

The life cycle of OnToology starts by visiting the tool landing page (see figure B-6). Then, the user enters the repository url and he will be redirected to GitHub website to privilege OnToology access to the repository. After that, OnToology will add a web-hook to the repository to listen to all future changes to the ontologies in this repository.

OnToology offers more features for logged in users. Users can login to OnToology through GitHub, which gives them access to "my repositories" page. They can see there repositories that are watched by OnToology and can "unwatch" a repository, so it won't be tracked for future changes. It also provide a nice interface to see OnToology configuration for each ontology in registered repositories and allows changing configuration of each directly (see figure B-7). This means, it will alter the configuration of the ontology in GitHub repository without creating a pull request.

5.2.2 GitHub

After registering the repository in OnToology, the repository will be tracked for changes. Once the user push any change to any of the ontologies in the registered repository, OnToology will be triggered and will generate documentation, diagrams and evaluation report for all ontologies changed in this repository. It will also create an issue in GitHub with the summary of ontology evaluation. The generation of documentation, diagrams and evaluation report can be enabled/disabled by a configuration file for each ontology (see 5.2.3). Finally, OnToology create a pull request with the generated files inside (see 5.3)

5.2.3 Configuration

OnToology create a default configuration file named "OnToology.cfg" for each ontology changed during a GitHub push of a registered repository if it doesn't have the configuration file. The configuration file allows enabling/disabling tools that will be called each time a change is pushed into GitHub. The configuration file have three sections, "ar2dtool" (for drawing diagrams), "widoco" (for documentations) and "oops" (for evaluation), each has a key called "enable", which can be either "true" or "false" (see B-5).

5.3 Output Structure

When Ontology is triggered by a change in a repository, it creates a folder called "OnToology" in the top level of the repository. All files generated by OnToology goes inside this folder. Next, OnToology will create for each ontology file that is changed (since last push) a folder we refer to it here as *ontology output folder*.

Each *ontology output folder* holds (OnToology) generated files for an ontology. *Ontology output folder* is also named after the ontology file it holds the generated files for. Moreover, *ontology output folder* will have the same path as the ontology path with "OnToology" added to the beginning of the path of *ontology output folder* (see a repository example before OnToology is triggered figure B-2 and after OnToology is triggered figure B-3).

Going deeper inside *ontology output folder*, OnToology will create three folders inside *ontology output folder* named "documentation", "diagrams" and "evaluation" and a configuration file "OnToology.cfg". The "documentation" and "evaluation" folders contain files generated by Widoco and OOPS, respectively. The "diagrams" folder contains three folders named "ar2dtool-class", "ar2dtool-taxonomy" and "config". The first one contains class diagram of the ontology and the second one contains the taxonomy diagram of the ontology. The last one has two configuration files for ar2dtool, one is class diagram configuration and the second is taxonomy diagram configuration.

5.4 Limitations

1. Repositories belong to GitHub organizations are not supported(due to GitHub limitation).
2. GitHub is the only supported version control.
3. Does not support OWL functional-style syntax¹².
4. Only works with the default GitHub branch 'master'.
5. Ontology Evaluation does not work with very large ontologies (due to OOPS! web-service timeout)

¹²<http://www.w3.org/TR/owl2-syntax/>

Chapter 6

Evaluation

After the implementation phase, we need to evaluate the problems we mentioned earlier. To do so, we conduct an experiment.

6.1 Experiment Steps

The experiment to be done announcing the tool to the semantic-web community. The announcement done through an email to semantic-web mailing list. The email introduces OnToology and includes examples and a how-to guide . Then, we monitor OnToology for errors. Monitoring registered repositories is done through checking OnToology main page. OnToology main page contains the latest repositories in a table with the status of each. If a repository has a problem, it will show the summary of the error next to the repository in the table, under the status column.

6.2 Results and Discussion

In the first few hours after announcing OnToology, we saw repositories being registered in OnToology.

For most of the registered repositories, documentation, diagrams and evaluation reports were generated successfully. For the other few, there were two sets, the first set of ontologies didn't have documentations and diagrams generated, the second set didn't have evaluation reports generated.

For the first set, part of the ontologies containing errors, the others using unsupported format (OWL functional-style syntax). For the second set, they used huge ontologies. Even through the generation of documentations and diagrams were taking hours, they were generated. For evaluation report, we were using OOPS! web-service, which because of the large size of the ontologies, was taking too much time, which then caused it to timeout. This means, the connection to OOPS! is lost, and hence, OOPS! won't be able to deliver the results back to OnToology. OnToology created issues in GitHub for each ontology that didn't generate the expected output. It also created pull request with the generated files and ignored the missing ones.

We got positive responses like "Performed beautifully" and "it passed with flying colors!", which makes a good impression that OnToology reached its goal. What confirm it more, is the reply we got from a W3C member who liked the tool and said that he is going to submit it to W3C.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we presented OnToology, an online tool to generate documentation and perform ontology evaluation. We mentioned problems in the current situation for ontology documentation and evaluation. We mentioned that the tools for ontology documentation and generation are not integrated with each other and they are executed separately. We also mention that ontology documentation and evaluation does not happen collaboratively. Next, we presented how OnToology tried to solved these problems. After that, we showed the success of OnToology in solving the problems as one of the users commented "passed with flying colors!".

7.2 Future Work

The novelty of OnToology opens doors for future work and extending the current state of the tool. One of the things we propose as future work is to add ontology publishing functionality. Integrate with other version control systems is also a nice thing to have

in the near future. We also received a suggestion from the community to integrate OnToology with GitLab, as they have their ontologies hosted there. Overcome the limitation we currently have in OnToology like supporting other Git branches beside the 'master' branch and support repositories belong to GitHub organizations can also increase the number of ontology developers benefiting from OnToology.

Appendix A

Tables

¹This table is reference is [47]

²This table is reference is [47]

Table A.1: VOWL2 Color Scheme

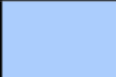

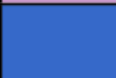
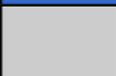
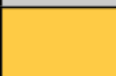


Name	Color	Application
General		classes, object properties, disjoints
Rdf		elements of RDF and RDF Schema
External		external classes and properties
Deprecated		deprecated classes and properties
Datatype		datatypes, literals
Datatype property		datatype properties
Highlighting		highlighted elements

Table A.2: Evaluation Tools Comparison¹

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

Table A.3: Pitfalls Categorization²

Human understanding	Modelling issues
<ul style="list-style-type: none"> • P1. Creating polysemous elements • P2. Creating synonyms as classes • P7. Merging different concepts in the same class • P8. Missing annotations • P11. Missing domain or range in properties • P12. Missing equivalent properties • P13. Missing inverse relationships • P19. Swapping intersection and union • P20. Misusing ontology annotations • P22. Using different naming criteria in the ontology 	<ul style="list-style-type: none"> • P2. Creating synonyms as classes • P3. Creating the relationship “is” instead of using "rdfs:subClassOf", "rdf:type" or "owl:sameAs" • P4. Creating unconnected ontology elements • P5. Defining wrong inverse relationships • P6. Including cycles in the hierarchy • P7. Merging different concepts in the same class • P10. Missing disjointness • P17. Specializing too much a hierarchy • P11. Missing domain or range in properties • P12. Missing equivalent properties • P13. Missing inverse relationships • P14. Misusing "owl:allValuesFrom" • P15. Misusing “not some” and “some not” • P18. Specifying too much the domain or the range • P19. Swapping intersection and union • P21. Using a miscellaneous class • P23. Using incorrectly ontology elements • P24. Using recursive definition • P25. Defining a relationship inverse to itself • P26. Defining inverse relationships for a symmetric one • P27. Defining wrong equivalent relationships • P28. Defining wrong symmetric relationships • P29. Defining wrong transitive relationships
Logical consistency	
<ul style="list-style-type: none"> • P5. Defining wrong inverse relationships • P6. Including cycles in the hierarchy • P14. Misusing "owl:allValuesFrom" • P15. Misusing “not some” and “some not” • P18. Specifying too much the domain or the range • P19. Swapping intersection and union • P27. Defining wrong equivalent relationships • P28. Defining wrong symmetric relationships • P29. Defining wrong transitive relationships 	
Real world representation	
<ul style="list-style-type: none"> • P9. Missing basic information • P10. Missing disjointness 	

Appendix B

Figures

1. Do you think it would be helpful if there is a tool to automate part of the ontology development process?

☐ Yes

☐ No

and Why?

2. What do you think if the tool automate the production of documentation, diagrams and evaluation report?

☐ Good Idea

☐ Bad Idea

and Why?

3. What if the tool also works in collaborative environment, do you think it would be helpful?

☐ Yes

☐ No

and Why?

4. Do you think the tool would increase the productivity, that it automates part of the work done by the developer?

☐ Yes

☐ No

and Why?

5. Do you think having the evaluation report will help detecting the pitfalls of the ontology in early stage?

☐ Yes

☐ No

and Why?

6. Do you think producing evaluation report of the ontology automatically during the development process will help produce ontologies with fewer error syntactically and semantically?

☐ Yes

☐ No

and Why?

7. Suppose that you have your ontology in a Github repository and there is a tool that will automatically generate documentation, diagrams and evaluation report of your ontology (every time you made a ~~push~~ push to the repository). What do you think if the documentation, diagrams and evaluation report is added to your repository through a pull request (rather than a direct push)? ☐ Having a pull request with documentation, diagrams and evaluation is a good idea

☐ I prefer direct push or other ways (please specify)

and Why?

8. Suppose you have the tool (mentioned in previous question), do you prefer to have a configuration file to enable/disable the generation of any of the following activities: documentation, diagrams, evaluation.? ☐ Yes
☐ No
and Why?
9. What if the configuration (mentioned in previous question) can also be change through a webpage, do you think it is a good idea? ☐ Yes
☐ No
and Why?

Figure B-1: Questionnaire



Figure B-2: Example of Repository Structure Before OnToology Triggering



Figure B-3: Example of Repository Structure After OnToology Triggering

```

<?xml version="1.0" encoding="UTF-8"?>
<OOPSRequest>
<OntologyUrl></OntologyUrl>
<OntologyContent>
<!-- This example is taken from
           http://www.linkeddatatools.com/introducing-rdfs-owl
-->
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

  <!-- OWL Header Example -->
  <owl:Ontology
    rdf:about="http://www.linkeddatatools.com/plants">
  <dc:title>
    The LinkedDataTools.com Example Plant Ontology
  </dc:title>
  <dc:description>An example ontology written for the
    LinkedDataTools.com RDFS and OWL introduction tutorial
  </dc:description>
  </owl:Ontology>

  <!-- OWL Class Definition Example -->
  <owl:Class
    rdf:about="http://www.linkeddatatools.com/plants#planttype">
    <rdfs:label>The plant type</rdfs:label>
    <rdfs:comment>The class of plant types.</rdfs:comment>
  </owl:Class>
</rdf:RDF>

  </OntologyContent>
  <Pitfalls></Pitfalls>
  <OutputFormat></OutputFormat>
</OOPSRequest>

```

Figure B-4: Request Example to OOPS! RESTful Web Service

```
[ar2dtool]
enable = false
[widoco]
enable = true
[oops]
enable = true
```

Figure B-5: A Sample configuration file

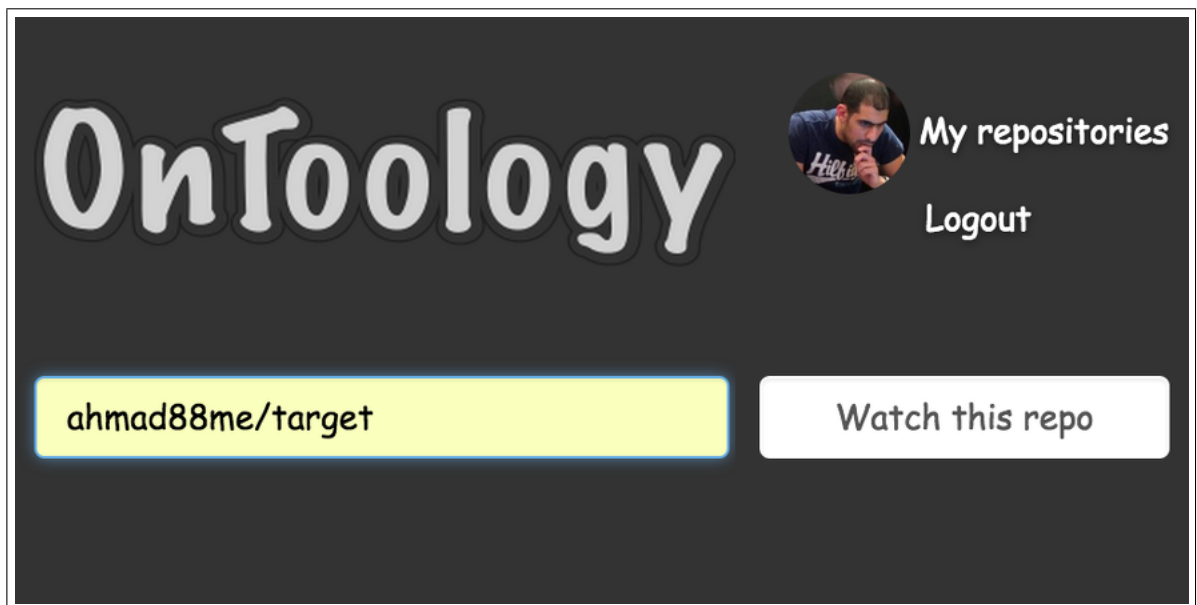


Figure B-6: OnToolology page to register repositories

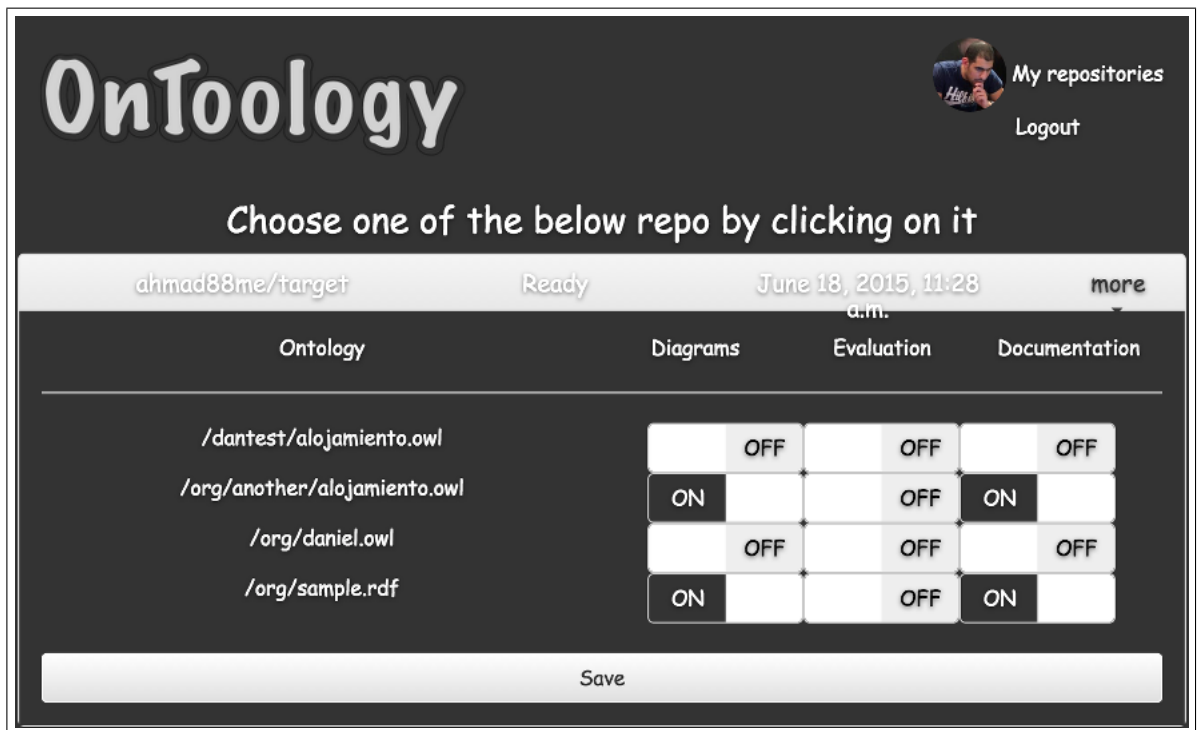


Figure B-7: OnToology configuration example

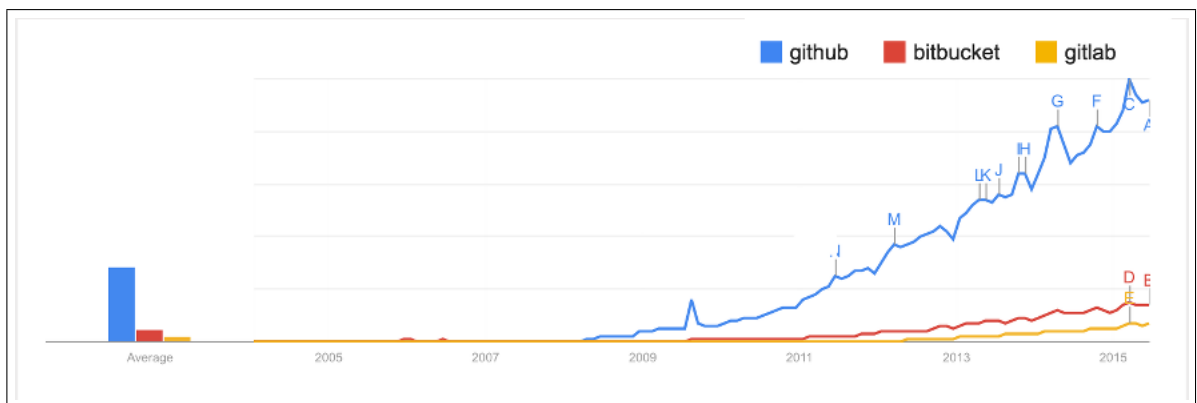


Figure B-8: Google trends comparison between Github vs Bitbucket vs GitLab[8]

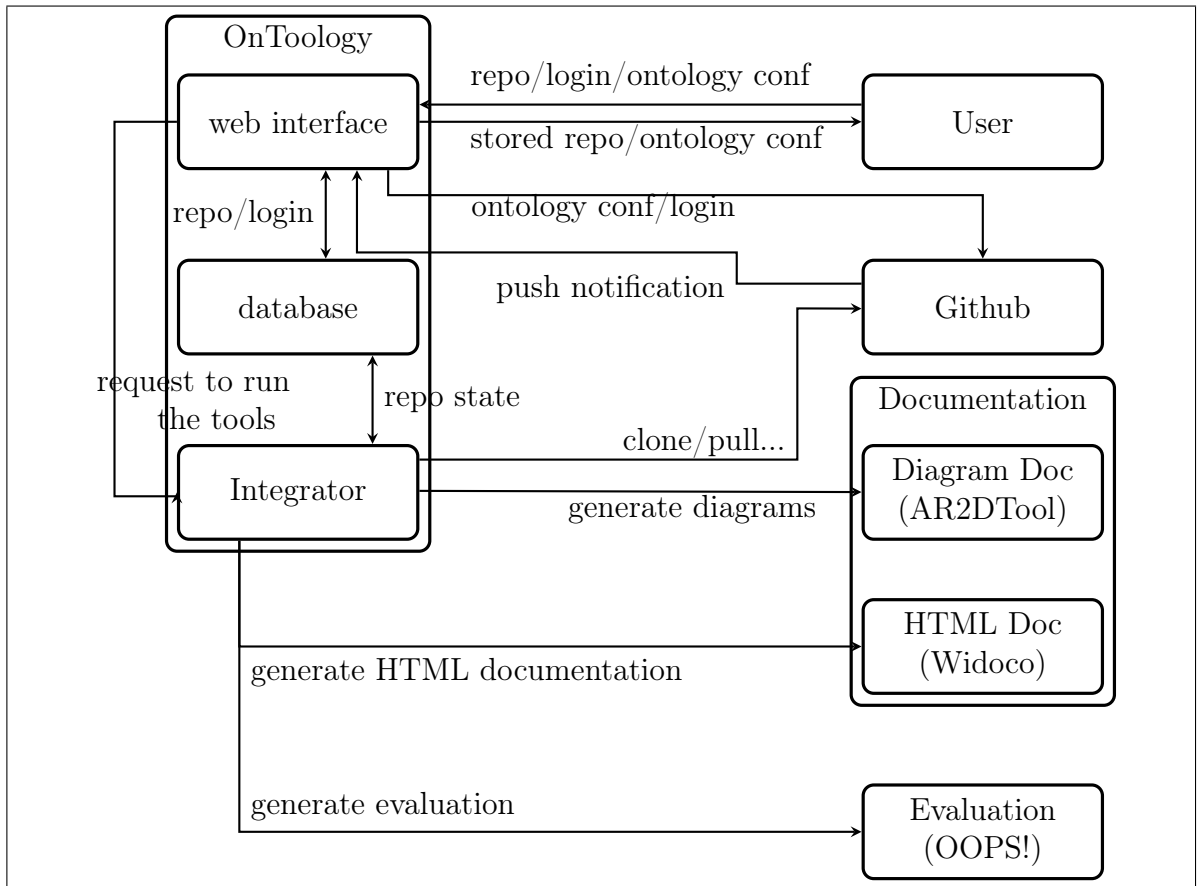


Figure B-9: OnToology Architecture

Bibliography

- [1] About version control. Available at <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control> (2015/07/13).
- [2] Generating ssh keys. Available at <https://help.github.com/articles/generating-ssh-keys/> (2015/07/19).
- [3] Git. Available at <https://git-scm.com/> (2015/07/12).
- [4] Git branching. Available at <https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell> (2015/07/13).
- [5] Github. Available at <https://en.wikipedia.org/wiki/GitHub> (2015/07/12).
- [6] Github glossary. Available at <https://help.github.com/articles/github-glossary/> (2015/07/13).
- [7] Github permission levels. Available at <https://help.github.com/articles/permission-levels-for-a-user-account-repository/> (2015/07/13).
- [8] Google trends: Github vs bitbucket vs gitlab. Available at <http://www.google.com/trends/explore#q=GitHub,Bitbucket,GitLab> (2015/07/1).
- [9] Oauth. Available at <https://developer.github.com/v3/oauth/> (2015/07/19).
- [10] Ontology design patterns. Available at <http://ontologydesignpatterns.org/wiki/Category:OntologyDesignPattern> (2015/07/11).
- [11] OOPS! pitfalls catalogue. Available at <http://oops.linkeddata.es/catalogue.jsp> (2015/07/11).
- [12] Parrot. Available at <http://ontorule-project.eu/parrot> (2015/07/12).
- [13] Parrot help. Available at <http://ontorule-project.eu/parrot/help> (2015/07/12).

- [14] Parrot wiki. Available at <https://bitbucket.org/fundacionctic/parrot/wiki/Home> (2015/07/12).
- [15] Pygithub repository. Available at <https://github.com/pygithub/pygithub> (2015/06/01).
- [16] Research methodology. Available at <http://www.fao.org/docrep/w3241e/w3241e05.htm> (2015/06/21).
- [17] Vocabularies. Available at <http://www.w3.org/standards/semanticweb/ontology> (2015/07/18).
- [18] Webprotege developer's guide. Available at <http://protegewiki.stanford.edu/index.php?title=%20WebProtegeDevelopersGuide&oldid=12158> (2015/05/22).
- [19] Program on survey research. *Harvard University*, 2007.
- [20] H. Alani. TGVizTab: An ontology visualisation extension for protégé. *In 2nd Workshop on Visualizing Information in Knowledge Engineering*, 2003.
- [21] Ahmad Alobaid, Daniel Garijo, María Poveda-Villaloñ, Idafen Santana-Perez, and Oscar Corcho. Ontoology, a tool for collaborative development of ontologies. *International Conference on Biomedical Ontology*, 2015.
- [22] Annika Öhgren. *Towards an Ontology Development Methodology for Small and Medium-sized Enterprises*. PhD thesis, Jönköping University, 2009.
- [23] Tim Berners-Lee. Cool URIs don't change. Available at <http://www.w3.org/Provider/Style/URI.html.en> (2015/06/04).
- [24] Eva Blomqvist, Valentina Presutti, Enrico Daga, and Aldo Gangemi. Experimenting with extreme design. *Proceedings of 17th international conference, EKAW*, 2010.
- [25] M. Bostock, V. Ogievetsky, and J. Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301-2309, 2011.
- [26] S. M. Falconer, C. Callendar, and M.-A. Storey. A visualization service for the semantic web. *In Proceedings of the 17th International Conference on Knowledge Engineering and Management by the Masses*, 2010.

- [27] Achille Fokoue, Aaron Kershenbaum, Li Ma, Edith Schonberg, and Kavitha Srinivas. The summary abox: Cutting ontologies down to size. *Proceedings of the 5th international conference on The Semantic Web*, 2006.
- [28] Daniel Garijo. Widoco. Available at <https://github.com/dgarijo/Widoco> (2015/06/05).
- [29] Chiara Ghidini, Marco Rospocher, and Luciano Serafini. Moki: a wiki-based conceptual modeling tool. *In Proceedings of ISWC Posters and Demonstrations Track*, 2010.
- [30] Paul Groth and Luc Moreau. An overview of the prov family of documents. Available at <http://www.w3.org/TR/2012/WD-prov-overview-20121211/> (2015/07/12), 2012.
- [31] Matthew Horridge and Peter F. Patel-Schneider. OWL 2 web ontology language manchester syntax. , 2009.
- [32] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosz, and Mike Dean. Swrl: A semantic web rule language combining OWL and RuleML. 2004.
- [33] A. Hussain, K. Latif, A. Rextin, A. Hayat, and M. Alam. Scalable visualization of semantic nets using power-law graphs. *Applied Mathematics and Information Sciences*, 2014.
- [34] J.Heer, S.K.Card, and J.A.Landay. Prefuse: A toolkit for interactive information visualization. *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '05, pages 421-430. ACM*, 2005.
- [35] Michael Kifer and Harold Boley. Rif overview. Available at <http://www.w3.org/TR/rif-overview/> (2015/07/12), 2013.
- [36] S. Krivov, R. Williams, and F. Villa. Gowl: A tool for visualization and editing of owl ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2007.
- [37] Piotr Kunowski and Tomasz Bojarski. Sova. Available at <http://protegewiki.stanford.edu/wiki/SOVA> (2015/07/05).
- [38] T. Liebig and O. Noppens. Ontotrack: A semantic approach for ontology authoring. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2005.

- [39] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. VOWL 2: User-Oriented Visualization of Ontologies. *Knowledge Engineering and Knowledge Management Lecture Notes in Computer Science*, 8876, 2014.
- [40] E. Motta, P. Mulholland, S. Peroni, M. d’Aquin, J. M. Gomez-Perez, V. Mendez, and F. Zablith. A novel approach to visualizing and navigating ontologies. *In Proceedings of the 10th International Conference on the Semantic Web*, 1, 2011.
- [41] F. J. García-Peñalvo, R. Colomo-Palacios, J. García, and R. Theroñ. Towards an ontology modeling tool. a validation in software engineering scenarios. *Expert Systems with Applications*, 2012.
- [42] Viktoria Pammer, Chiara Ghidini, Marco Rospocher, Luciano Serafini, and Stefanie Lindstaedt. Automatic support for formative ontology evaluation. *In Proceedings of EKAW Poster and Demo Track*, 2010.
- [43] Idafen Santana Pérez. Ar2dtool. Available at <https://github.com/idafensp/ar2dtool> (2015/06/06).
- [44] Silvio Peroni, David Shotton, and Fabio Vitali. Latest developments to lode. *Knowledge Engineering and Knowledge Management Lecture Notes in Computer Science Volume 7603*, pp 417-420, 2012.
- [45] Silvio Peroni, David Shotton, and Fabio Vitali. Making ontology documentation with lode. *Proceedings of I-SEMANTICS Posters and Demonstrations Track*, 2012.
- [46] Niklas Petersen, Lavdim Halilaj, Christoph Lange, and Sören Auer. Vocol: An agile methodology and environment for collaborative vocabulary development. , 2015.
- [47] María Poveda-Villalón, Mari Carmen Suárez-Figueroa, and Asunción Gómez-Pérez. Validating ontologies with oops! *18th International Conference on Knowledge Engineering and Knowledge Management*, 2012.
- [48] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 2007.
- [49] M.-A. Storey, N. F. Noy, M. Musen, C. Best, R. Ferguson, and N. Ernst. Jambalaya: An interactive environment for exploring ontologies. *In Proceedings of the 7th International Conference on Intelligent User Interfaces*, 2002.

- [50] Carlos Tejo-Alonso, Diego Berrueta, Luis Polo, and Sergio Fernández. Metadata for web ontologies and rules: Current practices and perspectives. *Metadata and Semantic Research Communications in Computer and Information Science Volume 240*, pp 56-67, 2011.
- [51] Tania Tudorache, Csongor Nyulas, Natalya F. Noy, and Mark A. Musen. Webprotégé: A collaborative ontology editor and knowledge acquisition tool for the web. *Semantic Web*, 2013.