

Web-based Ontology Browser with Advanced Analysis Features

A dissertation submitted to The University of Manchester for the degree of Master in Advanced Computer Science in the Faculty of Engineering and Physical Sciences

2016

**Ghadah Abdulrahman S Alghamdi
ID 9546842**

School of Computer Science

Table of Contents

Table of Contents.....	2
List of Figures.....	4
List of Tables	7
Abstract.....	8
Declaration	9
Intellectual Property Statement	10
Acknowledgment.....	11
1 Introduction	12
1.1 Project Introduction and Motivation.....	12
1.2 Project Objectives and Scope	13
1.3 Report Structure.....	14
2 General Background.....	15
2.1 Introduction.....	15
2.2 Description Logics.....	16
2.3 OWL Ontologies.....	18
2.4 Related Approaches	21
2.5 Summary.....	25
3 Used Tools.....	26
3.1 LETHE.....	26
3.2 OWL API.....	27
3.3 VOWL	29
3.4 Summary.....	33
4 Design and Implementation	34
4.1 Requirements and Overall System Architecture	34
4.2 Implementation Platforms and Programming Language	37
4.3 Use Cases' Implementation	43
4.4 System Integration	49
4.5 Summary.....	51

5	System Testing and Illustration	52
5.1	System Testing.....	52
5.2	LETHE Web-Analyser: System Illustration	55
5.3	Summary.....	65
6	Case Studies	66
6.1	Uniform Interpolation and Forgetting Case Study.....	66
6.2	Logical Differences Case Study	84
6.3	VOWL and OntoGraf Comparison.....	91
6.4	Our Tool and PATO Tool Comparison.....	94
6.5	Summary.....	99
7	Conclusions and Future Work	100
	References	104

List of Figures

Figure 2-1: The structure of a knowledge representation system based on DLs	16
Figure 2-2: Illustration of Ontology Class inheritance Structure	20
Figure 2-3: Classes and their individuals connected by properties	20
Figure 2-4: OWLViz visualisation of the travel ontology	24
Figure 3-1: The UML class diagram of OWL API main interfaces.....	28
Figure 3-2: The difference between owl:Class and rdfs:Class representation.....	30
Figure 3-3: Illustration of VOWL visual elements	30
Figure 3-4: Tables (a) and (b) illustrate VOWL graphical primitives and the colour schema respectively	32
Figure 4-1: The system architecture represented as modules in the client and server sides..	35
Figure 4-2: Illustration of the system's request processing workflow in Spring MVC	41
Figure 4-3: Code snippet of the Tiles configuration file.....	42
Figure 4-4: Code of handleRequest method in uniform interpolation controller class.....	43
Figure 4-5: Code of uploadFile method in OntologyFile class	44
Figure 4-6: Code snippet for populating ontology symbols to the user.....	44
Figure 4-7: Code snippet for converting an ontology to a JSON file	45
Figure 4-8: Code snippet for passing a JSON file name to the view layer.....	46
Figure 4-9: Illustration of webVOWL visualising the "test-ontology-iri-4" file	46
Figure 4-10: Code for ALCH TBoxes interpolation in uniform interpolation class.....	47
Figure 4-11: Code for the method that computes logical differences based on the ALCH TBoxes forgetting method (common symbols)	47
Figure 4-12: Code for the method that computes the logical differences based on the ALCH TBoxes forgetting method (specified symbols).....	48
Figure 4-13: Code for the method that saves ontologies computed by uniform interpolation	48
Figure 4-14: Code for the method that saves ontologies computed by logical differences...	49
Figure 4-15: The system integration using Nginx server between Tomcat and NodeJs servers	51
Figure 5-1: Illustration of tests under running in IntelliJ performed on LETHE_web	53
Figure 5-2: Main Page of LETHE Web Analyser	55
Figure 5-3: The overall page of uniform interpolation interface	56
Figure 5-4: First Step (upload ontology) in uniform interpolation page	57
Figure 5-5: Steps 2 (select mode) and 3 (select forgetting method) in uniform interpolation page.....	58

Figure 5-6: Step 4 (select symbol) in uniform interpolation page	58
Figure 5-7: Symbols filtration in Step 4 in uniform interpolation page	59
Figure 5-8: Resulting ontology after the selection of symbols in uniform interpolation page	59
Figure 5-9: (Visualise) and (Download as OWL/XML) buttons located under resulting ontology box in uniform interpolation page	59
Figure 5-10: Ontology information section and fifth step (further computation) in uniform interpolation page	60
Figure 5-11: The overall page of logical differences interface	62
Figure 5-12: Metadata subsection of webVOWL interface	63
Figure 5-13: The overall page of webVOWL interface showing the visualisation of bibtex ontology	64
Figure 6-1: The resulting ontology based on ALCH TBoxes forgetting method (uniform interpolation mode).....	68
Figure 6-2: Some of the travel original ontology axioms	68
Figure 6-3: webVOWL visualisation of the resulting ontology based on ALCH TBoxes forgetting method (uniform interpolation mode).....	69
Figure 6-4: The resulting ontology based on ALCH TBoxes forgetting method after including object properties (uniform interpolation mode).....	70
Figure 6-5: The resulting ontology based on SHQ TBoxes forgetting method (uniform interpolation mode).....	71
Figure 6-6: webVOWL visualisation of the resulting ontology based on SHQ TBoxes forgetting method (uniform interpolation mode).....	72
Figure 6-7: The resulting ontology based on ALC with ABoxes forgetting method (uniform interpolation mode).....	73
Figure 6-8: webVOWL visualisation of the resulting ontology based on ALC with ABoxes forgetting method (uniform interpolation mode).....	74
Figure 6-9: The resulting ontology based on ALCH TBoxes forgetting method (forgetting mode).....	77
Figure 6-10: webVOWL visualisation of the resulting ontology based on ALCH TBoxes forgetting method (forgetting mode)	78
Figure 6-11: The resulting ontology based on SHQ TBoxes forgetting method (forgetting mode).....	79
Figure 6-12: webVOWL visualisation of the resulting ontology based on SHQ TBoxes forgetting method (forgetting mode)	80
Figure 6-13: The resulting ontology based on ALC with ABoxes forgetting method (forgetting mode).....	81

Figure 6-14: webVOWL visualisation of the resulting ontology based on ALC with ABoxes forgetting method (forgetting mode)	82
Figure 6-15: SWRC Ontology main concepts structure	85
Figure 6-16: webVOWL visualisation of the SWRC ontology version 0.3	87
Figure 6-17: webVOWL visualisation of the SWRC ontology version 0.7.1	88
Figure 6-18: The resulting ontology after applying logical differences function.....	89
Figure 6-19: Part of the visualisation of the SWRC old version ontology showing the Unpublished class is a subclass of Publication.....	90
Figure 6-20: OntoGraf visualisation of the travel ontology	92
Figure 6-21: The interface of PATO illustrating the step of inserting an ontology and the .net file before the conversion process	95
Figure 6-22: Pajek visualisation of the resulting .clu (partition) file, illustrating the produced modules in different colours	97
Figure 6-23: wevVOWL visualisation of the resulting restricted view ontology.....	98
Figure 6-24: Extract of .net file syntax.....	98
Figure 6-25: The resulting ontology in readable format.....	98

List of Tables

Table 1: Summary of the unit tests performed on LETHE_web	53
Table 2: Summary of the print statements tests performed on LETHE_web	54
Table 3: Summary of the resulting axioms of all forgetting methods (uniform interpolation mode).....	75
Table 4: Summary of the resulting axioms of all forgetting methods (forgetting mode).....	83
Table 5: Summary of the resulting axioms of logical differences function applied on the SWRC ontology.....	90
Table 6: Summary of the comparison between webVOWL and OntoGraf	94
Table 7: Summary of the comparison between PATO and our tool	99
Table 8: Summary of the features provided by the LETHE standalone version and the LETHE web version	101

Abstract

Ontologies in information science are representation of certain domain concepts. They are used in knowledge representation systems that include areas such as medicine or engineering. These ontologies can be too large in order to fit all of an application's vocabularies. Thus, there has been an increasing interest in producing smaller modular views of ontologies that might otherwise be too big to explore and analyse. These approaches include modularisation based on partitioning methods and module extraction, which assists analysis, inspection and use for various purposes by producing modules from larger ontologies. However, the resulting modules are likely to be semantically weak, with conceptual redundancies.

LETHE is an implementation of uniform interpolation, logical differences and TBox abduction, supporting expressive description logics, which are languages that express concepts in a structured way. **LETHE** provides restricted views of ontologies based on saturation-based reasoning that helps to preserve the logical entailments of the smaller ontologies. This has many applications, including ontology analysis, information hiding and ontology reuse. The aim in this project is to develop a web ontology analyser with advanced analysis features based on the **LETHE** tool. The features include visualisation capabilities beneficial for users other than computer scientists or ontology developers, helping these users to gain deep understanding of ontologies and their characteristics. Ontology developers can exploit the analyses features to focus on selected details of an ontology view, which can be useful for debugging purposes.

The principal aim is to deliver an ontology analyser that supports the functionalities of **LETHE**, exploiting visualisation features provided by VOWL. Development of the tool involves the use of the OWL API, making it possible to achieve **LETHE** functionalities within the ontology analyser.

Declaration

No portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Intellectual Property Statement

- i. The author of this dissertation (including any appendices and/or schedules to this dissertation) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.
- ii. Copies of this dissertation, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has entered into. This page must form part of any such copies made.
- iii. The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the dissertation, for example graphs and tables (“Reproductions”), which may be described in this dissertation, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.
- iv. Further information on the conditions under which disclosure, publication and commercialisation of this dissertation, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the [University IP Policy](#), in any relevant Dissertation restriction declarations deposited in the University Library, and The [University Library’s regulations](#).

Acknowledgment

First, I would like to express my sincere gratitude to my parents who have given me love and support during my time in Manchester. Their continued communication with me despite the long distance has encouraged me to work harder. This accomplishment would not have been possible without them. Thank you.

Second, I would like to thank my supervisor, Dr Renate Schmidt, for her consistent support and professional advice during the development of the project. She was always available any time I needed help or advice. Thank you.

Next, I would like to thank Dr Patrick Koopmann for his help in some of the project matters. Thank you for your quick responses to my questions.

Finally, I would like to express my gratefulness for the opportunity that I had to pursue my master's degree at the University of Manchester. It was such a wonderful experience.

1 Introduction

This chapter aims to provide the introduction of the project and its motivation, objectives and scope and structure of the report.

1.1 Project Introduction and Motivation

In computer science, ontologies provide terminological information that can be used in knowledge representation systems. This information represents certain domain vocabularies, describing the concepts taxonomies and the relationships between them. The representation of domain concepts and vocabularies is called ontology. Ontologies importance lies in their usage in artificial intelligence applications, as they are the core of such applications. Knowledge representation systems have many applications in areas that include medicine, bio-informatics, the semantic web and software engineering, but analysis tools and editors are needed to enable users to understand the information provided by potentially large ontologies. Tools such as Protégé, SWOOP or TopBraid offer many such functions, including graphical representation of ontologies, reasoning services, explanation of findings and refactoring methods [1]. Some of these tools also enable segmentation of ontologies into modules to provide a restricted view. These include the Protégé plugin **Prompt**, which is described in more detail in Section 2.4. As such tools produce modules based on the syntactical properties of ontologies that are likely to include conceptual redundancies between modules, an analysis tool is needed that can produce restricted views of ontologies while preserving their semantics. Here, **LETHE** is used for this purpose.

LETHE, which was developed by Patrick Koopmann, is based on expressive description logics [2]. Description logics provide models consisting of concepts (or classes), roles, individuals and the relationships between them [3]. **LETHE** supports three expressive description logics: Attributive Language with Complements (ALC), Attributive Language with Complement and Role Hierarchies (ALCH) and ALC extended with transitive role axioms with quantified number restrictions (SHQ) [2]. The core functionality of **LETHE** is the computation of uniform interpolation of ontologies, extracting specific concepts of an ontology based on user selection of a particular sub-signature. This process yields a restricted

view of the ontology with several potential applications, such as ontology analysis or information hiding [3]. In addition, the tool performs two other functionalities: logical differences and TBox abduction. Logical differences seek to produce the different entailments of axioms of two ontologies [3] and TBox abduction aims to add a set of axioms of desired entailments to a given ontology [3].

1.2 Project Objectives and Scope

The aim of the present project is to develop a web browser with advanced analysis features that exploits the capabilities of the **LETHE** tool, allowing the user to create a restricted view of ontologies of interest to them. In addition, through the visualising functionality, the application will enable the user to gain a deep understanding of the logical structure of ontologies and the relationships between terms. Ontology developers can also exploit this tool to focus on details that might be less accessible in the large (original) ontology, assisting analyses of the ontology. **LETHE** is used as the backend of this web browser to achieve the following objectives.

- Develop a web application to provide visualisation capabilities for ontologies.
- Investigate the capabilities of the **LETHE** Java library in different types of ontology applications.
- Integrate the **LETHE** Java library with the web application to achieve the library's functionalities in an efficient way.
- Make the application a graphical mirror for what is implemented in the backend of **LETHE**.
- Determine the sizes of ontologies that **LETHE** can reasonably handle and identify bottlenecks.
- Provide an artefact within the scope of the application (such as a manual) as a resource for learning how to use **LETHE**.
- Apply the analyser to a real-life ontology in medicine or bio-informatics.

1.3 Report Structure

This report is structured as follows: Chapter Two outlines the background to ontologies, description logics, OWL ontologies and the related approaches, which include modularisation and visualisation approaches. The third chapter describes the tools used during the development of the project, including **LETHE**, OWL API and VOWL. The fourth chapter describes the design of the system and the implementation methodology, clarifying the overall system architecture along with the platforms of the implementation. The system integration is also described in the chapter. Chapter Five presents the testing methods that were conducted on the tool along with an illustration of the system's functionalities. Chapter Six provides case studies performed to evaluate the system outcomes. Comparisons between the tools mentioned in the related approaches and our tool is presented in the chapter as well. Chapter Seven draws conclusions about the project and describes some objectives to be met in the future.

2 General Background

This chapter presents general background topics that are relative to the project. The chapter aims to define the scope of the project and provide reference to the concerned topics of the project. These topics include introduction to ontologies, description logics, OWL ontologies and related approaches. The related approaches section discusses other segmentation methods that lead to smaller ontologies. In addition, visualisation approaches that are used to provide graphical illustration of ontologies are discussed under the related approaches section.

2.1 Introduction

In computer science, ontologies are forms of data that represent information in a structured way through the use of certain tools. Gruber defined ontologies in [4] as “an explicit specification of conceptualisation”. According to his definition, ontologies consist of concepts that describe a certain area or topic relating to objects in our world [5]. These concepts are specified by their domain or field, the individuals that relate to such concepts and the relationships between them [5].

Descriptions of ontologies use logic languages such as Description Logics (DLs) or first-order logic (FOL), contributing to clearer and more precise descriptions of concepts [6]. They define concepts as axioms, consisting of classes that formalise the subparts of a concept belonging to a certain domain. For instance, teacher is a class that describes the concept of teachers’ existence, representing part of the larger concept of education. The use of logical languages also facilitates graphical representation of concepts for easier and more concise identification of their content [3].

Ontologies can be used in many different fields, including medicine, bio-informatics and engineering. In healthcare, ontologies have been used extensively to define concepts such as symptoms, diseases or drugs, which can be used in knowledge systems to search in a professional and efficient way for suitable treatments [7].

2.2 Description Logics

Description Logics (DLs) provide logical descriptions of ontology components, formalising concepts in a structured way to enable their use in knowledge representation systems [6]. Figure 2-1 illustrates the structure of such a system. In description logics, the knowledge base consists of concepts and roles. These knowledge bases focus on reasoning, helping to deduce implicit knowledge from the explicit knowledge in the knowledge base [8]. In addition, many information processing systems use inference patterns supported by DLs [8]. These patterns form conceptualisation of concepts and individuals of objects in the world, producing structures that define sub-concept and super-concept relationships between classes. Such structures form hierarchies, connecting different concepts and enabling fast processing through inference services [8]. Moreover, the concepts and individuals of a knowledge base can be deduced automatically through the use of inference procedures [8].

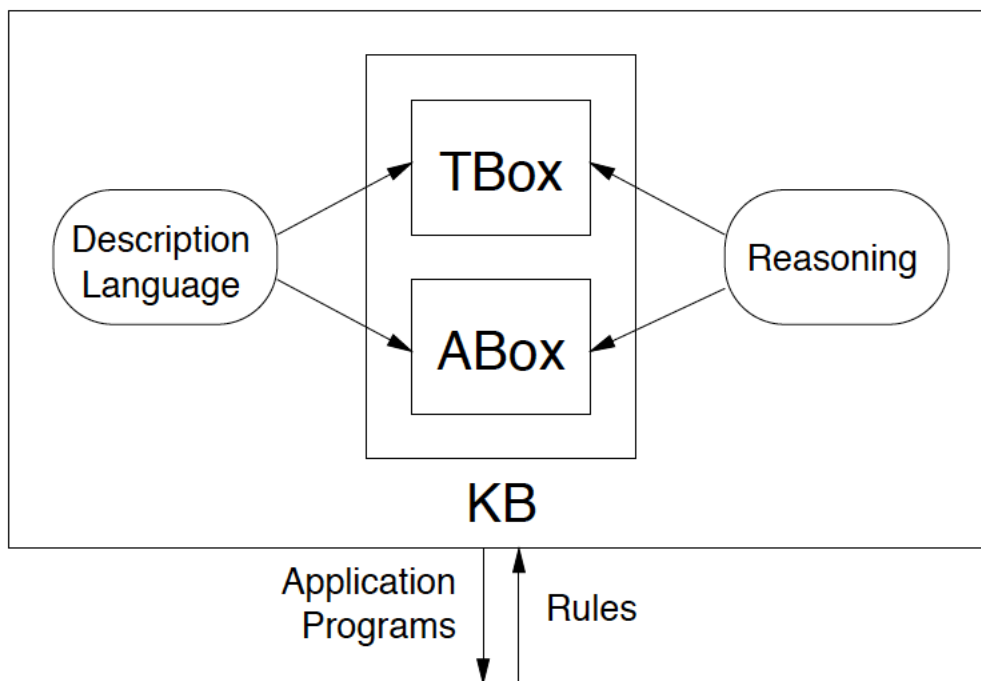


Figure 2-1: The structure of a knowledge representation system based on DLs [8]

In description logics, ontologies comprise of two parts: ABox and TBox [6]. The ABox represents knowledge of facts or assertions about individuals belonging to a specific concept [9]; for example, $(\text{Female} \cap \text{Person})(\text{Anna})$ means that there is a person who is female called Anna. This states the fact of the existence of such a person called Anna. On the other hand, The TBox provides terminological information about concepts and the relationships between them, describing the conceptualisation of a knowledge base [9]. For example: $\text{Woman} \equiv \text{Person} \cap \text{Female}$ describes the concept of a woman as a person who is female.

Moreover, Baader [8] defines a cyclic ontology as an ontology that has concepts which refers to itself. For example, A and B are atomic concepts that belong to an ontology O. If B is used directly by A, where B is on the right-hand side of an axiom of the form: $A \equiv \exists r.B$, $B \equiv s.A$, which its left-hand side is A. Then the ontology of this form is a cyclic ontology. Otherwise, the ontology is called acyclic [8]. For instance, the axiom

$$\text{Human}' \equiv \text{Animal} \cap \forall \text{hasParent.Human}'$$

It means that the *Human* class is equivalent to the concept of every animal who has a human parent is human. Here, the *Human* class directly uses the *Animal* class.

Description logics feature the ability to express complex types of knowledge by using expressive description logics, which are extended types of DLs. While there are many types of expressive description logics, four types in which three of them used by **LETHE** is referred to for the purposes of this report. The basic type is ALC (Attributive Language with Complements), which enables concepts to be described using a range of special types such as Universal concept (\top), of which every other concept is a sub-concept, and Bottom concept (\perp), which is a sub-concept of all other concepts, and of which no concept can be a sub-concept [8]. In addition, ALC defines a set of constructors: Negation ($\neg C$), Conjunction ($C \cap D$), Disjunction ($C \cup D$) and Existential quantifier ($\exists R.C$) [8]. These help to define concepts in an expressive and flexible way; for example, the axiom $\text{Professor} \sqsubseteq (\text{Person} \cap \text{UniversityEmployee}) \cup (\text{Person} \cap \neg \text{Student})$ translates as “There is a professor who is a person and who is also a university employee, or a person who is not a student”. The description logic ALCH [10] is an extension of ALC that includes role inclusion axioms; for example, $\text{parentOf} \sqsubseteq \text{ancestorOf}$ means that the role *parentOf* is a sub-role of the role *ancestorOf* [9].

A third type of description logic is SHQ, which is an extension of ALCH. It includes transitive roles, where roles can be extended to new roles [10], and qualified number restrictions, where a role can be restricted to a specified number of roles. Another expressive language of description logics is SHOIN. It involves a set of constructors that are more expressive including nominals, inverse roles and cardinality restrictions. The nominal constructor allows us to define individuals within axioms that are other than the ABox statements [8]. For example, the axiom $\text{Student} \cap \exists \text{hasCourse}.\{\text{mathematics}\}$, meaning that the students have at least one mathematics course. The inverse roles constructor helps to define a certain property to be inverted in the perspective of another one. For example, **hasChild** is the inverse of **hasParent** role [8]. The cardinality restrictions constructor lets us define number restrictions that exist in a concept. It is written as $\geq n R$ (at-least restriction) and $\leq n R$ (at-most restriction), where n is the range of nonnegative integers, and R refers to a role [8, 11]. For instance, the axiom $\text{Teacher} \cap \geq 2 \text{teaches.Subject}$ implies that a teacher teaches at least two subjects.

The structuring nature of DLs enables the construction of languages that are used to express ontologies for use in machine processing [12]. One of these languages is called OWL, which is described in more detail in Section 2.3. OWL is an extension of its predecessors, OIL and DAML+OIL [12, 13]. In the OWL language, *concepts* are called *classes*, and *roles* are called *properties*. As in DL, OWL uses constructors to create classes. For example, the negation of concept C ($\neg C$) has its opposite constructor in OWL, which is `complementOf`. Detailed description of other constructors can be found in [13].

2.3 OWL Ontologies

OWL (Web Ontology Language) is a semantic web language developed by the World Wide Web Consortium (W3C), enabling concepts of ontologies to be clearly described, using operators to describe complex axioms (concepts) such as intersection and union. These

operators enable new concepts to be both defined and described. The W3C has developed an extended second version of OWL, which is known as OWL 2 [14].

The main components of OWL ontologies are classes, properties and individuals.

- **Classes:** According to Horridge [17], classes are “sets that contain individuals”. This means that classes describe the information of a certain area or topic, containing individuals of concepts that belong to that particular class. Hierarchies can be defined for a specific topic, forming subclass-of or superclass-of relations; these hierarchies are known as *taxonomies* [17]. For example, in the class **Teacher**, which is a subclass of **Person**, **Teacher** can contain the set of individuals that define teachers. If teacher *Sami* is an individual belonging to the class **Teacher**, this implies that teacher *Sami* is also a person because **Teacher** is a subclass of the class **Person**. Each OWL ontology has a general (Top) class called **Thing**. All of the individuals that belong to a certain ontology are members of its **Thing** class. Figure 2-2 illustrates the hierarchical structure of classes in an ontology.
- **Individuals:** These represent instances or members of a class, defining objects of a certain topic.
- **Properties:** They identify relationships between individuals. For example, in the axiom *Joseph isTaughtBy Sami*, **isTaughtBy** is the property describing the information that student *Joseph* is taught by teacher *Sami*. In OWL 2, there are three types of properties: object properties, datatype properties and annotation properties [17].
 - Object properties define the relationships between two individuals. For example, the axiom *Sarah hasSibling Mary* states that *Sarah* has a sibling called *Mary*, where **hasSibling** is the object property.
 - Annotation properties define descriptions of ontologies that include the ontology’s properties, classes and individuals. They provide information such as the name of the ontology’s author and its date of creation. They do not introduce additional semantic meaning to the ontology [17].
 - Datatype properties add data values to individuals. For example, the **hasLastName** property would describe the information that *Sami*’s teacher has the last name *Stewart*.

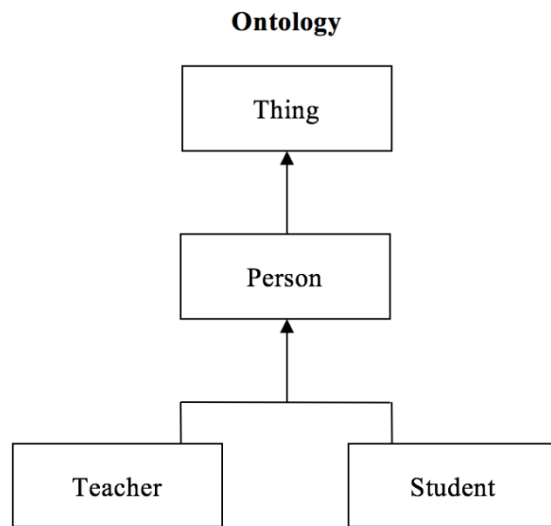


Figure 2-2: Illustration of Ontology Class inheritance Structure

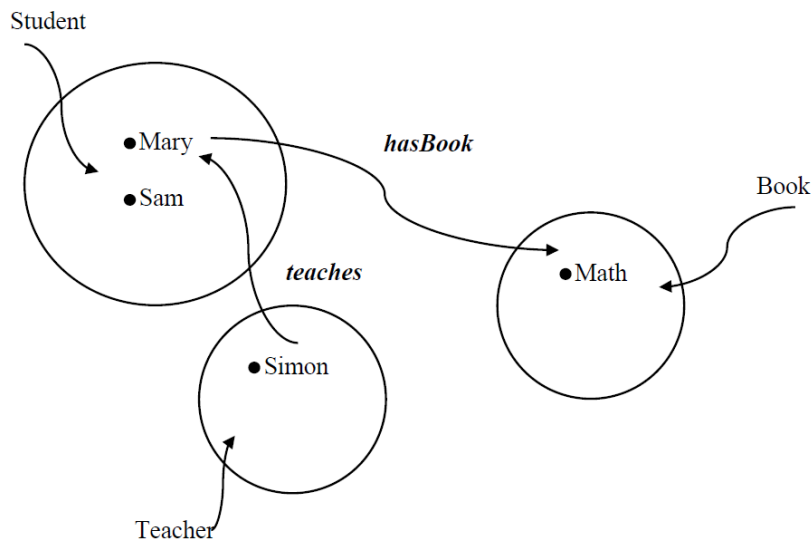


Figure 2-3: Classes and their individuals connected by properties

Figure 2-3 illustrates the set of classes **Teacher**, **Student** and **Book**, consisting of individuals such as *Mary* and *Sam* as part of the **Student** class. As individuals of the **Student** class, these represent two students. Individuals can also be connected with other individuals that belong to a different class through the use of properties. In this case, for instance, *Simon* is connected to *Mary* by the **teaches** property, which implies that teacher *Simon teaches* student *Mary*.

2.4 Related Approaches

2.4.1 Modularisation Approaches

Modularisation divides large ontologies into smaller ontologies or modules. This has many advantages, including easier maintenance of ontologies, improved efficiency of use in different applications and enhanced understanding of very large or complex ontologies [18].

This section discusses some approaches to modularisation and their implementation of tools for producing modules of ontologies. Partitioning approaches are discussed with reference to implementation of the **PATO** tool [19, 20], and module extraction is discussed with reference to **Prompt** [21].

In ontology partitioning, ontologies are divided into several modules by splitting the set of axioms that belong to a certain ontology into a set of modules $\{M_1, \dots, M_k\}$ where M_i is an ontology [18]. The original ontology can be reproduced by taking the union of all of its modules. Partitioning approaches vary in how they produce partitions, depending on their goals and desired results [18].

Structure-based partitioning [22] aims to produce modules with weak connections between them. The intent is to support the examination of small parts of ontologies, contributing to better maintenance [18]. The advantage of this approach is that its algorithms enable results to be matched to application requirements by providing the algorithm with appropriate parameters. However, this is a non-automatic approach to producing modules [18]. One example of this approach is the **PATO** tool [19], which is a standalone application that can be used to partition ontologies to produce small modules [18]. By accepting different parameters from the user at every step of the process, the result is tuned to the desired application requirements [18], using a semi-automatic mechanism to produce modules [19].

The other modularisation approach is module extraction approach (traversal view approach), which is based on recursive traversal of a set of related elements [18]. The recursion process uses a vocabulary selected from the original ontology to extract modules from it, taking an ontology O and a set of terms SV that belong to a certain signature of the ontology $SV \subseteq Sig(O)$ and returning M_{SV} as a module [18]. The approach is well illustrated by a tool called **Prompt**, a plugin for the popular ontology editor Protégé [23].

The **Prompt** suite comprises several packages, one of which is **PromptFactor**, whose functionality is to take sub-parts of ontologies. This relies on the mechanism of traversal view extraction, a functionality described in [21] as *factoring sub-ontologies* that finds inconsistencies between the terms (concepts) and list dangling references to the user [21]. One disadvantage of this tool is that the user needs deep understanding of the ontology to be worked on [18] because of the requirement to manually select classes and their properties. This makes it difficult for users to select classes from very large ontologies in order to produce modules [18].

The above tools perform modularisation by finding restricted (smaller) views of larger ontologies. Although the procedures used to produce modules are not expensive in terms of effort and processing time, they are likely to produce weak semantic modules. Moreover, user interaction with some of these tools involves complexity. For example, the **Prompt** tool requires users to interact at every step of the process. In addition, partitioning approaches produce modules based on syntactical properties of the ontologies without the use of reasoners, especially in the case of the **PATO** tool [18]. This is because these approaches extract modules using structure-based partitioning and may also inherit conceptual redundancies between modules from the original ontology [18]. Furthermore, these are standalone systems that cannot be used by the web. The following is a summary of the key advantages of our web-based tool for ontology analysis.

- Ability to select desired signatures to be included in the resulting ontology. The selection is performed manually in the current version. However, the future version will involve a semi-automatic way of selecting symbols.
- Production of smaller ontologies, based on the new saturation-based reasoning methods developed within **LETHE**.
- Provision of a deep visualisation tool for restricted view ontologies.
- Ease of use of in a web browser without having to install locally on the machine.
- Access on all operating systems, which is an advantage for both developers and users, who may wish to use different machines or different operating systems.

2.4.2 Visualisation Approaches

Browsing ontologies in a graphical representation is an important aspect of ontology understanding. Visualisation facilitates the process of analysing ontologies. There are many tools and plugins that were developed for such a purpose. Some of these tools are OntoGraf [24] and OWLViz [25]. Both of them are plugins to the popular ontology editor Protégé. Developed as collaboration between the University of Manchester and Stanford University, Protégé is an ontology editor that can be used to create ontologies for use in semantic web applications. It consists of many suites, including visualisation methods, used to analyse ontologies. In addition, the user can edit ontology files by uploading them to the editor [17].

OntoGraf is used to visualise ontologies and provide various visualisation features that can help non-expert users in the comprehension of ontologies. This tool went through several improvements through its development of many versions. Its latest available version is 2.0.3 which is compatible with Protégé-OWL 5.0. The key version of the tool is 0.0.1, which is the first version. This version included support of various layouts, the relationships between nodes and filtering them, and the ability to zoom the graph. Another important release of the tool is 0.0.3, in which tooltips feature was added, which can be configured to show or hide different detailed information. In addition, the capability of exporting the graph to image type was added in this version. Detailed information will be apparent from the comparison that is given in Section 6.3.

Another plugin of Protégé for visualising ontologies is OWLViz. It is used mainly to visualise class hierarchy relationships. OWLViz exploit the use of automated reasoning by providing an inferred hierarchy view along with the asserted hierarchy view. In addition, through the plugin we can distinguish the inconsistent concepts among other ones as they are highlighted in red. One disadvantage of OWLViz is that it is only restricted to visualising class hierarchies and ignores an essential type of ontology elements which are object properties. This can makes it difficult for non-expert ontology users to gain a proper idea about the ontology. It features the ability to export the graphs to PNG, JPEG or SVG files. Moreover, it supports several cognitive options that users can select including show class, show parents, show children, hide class, hide parents and hide children [26]. Figure 2-4 shows the travel ontology [49] visualised by OWLViz in the inferred model.

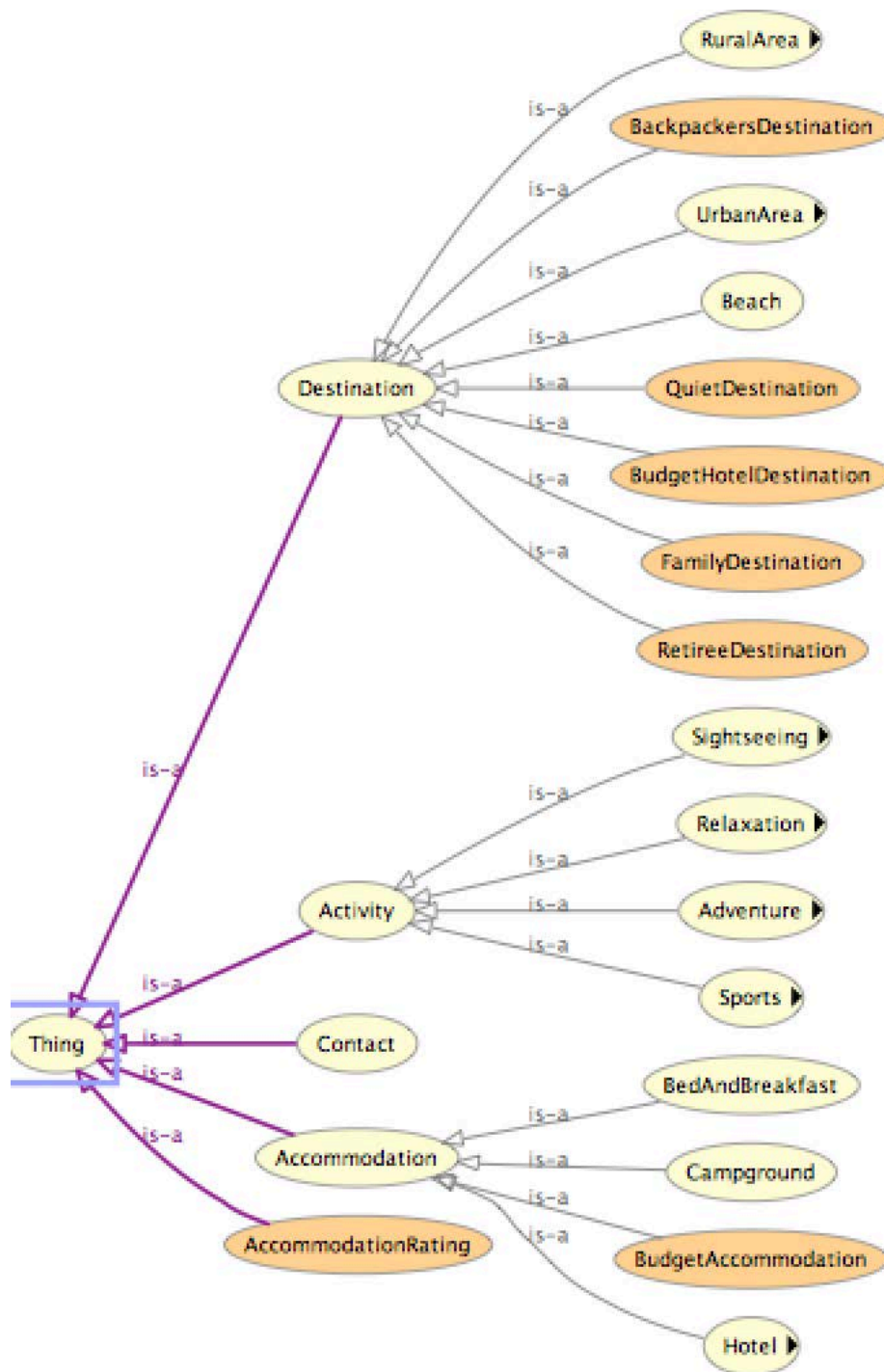


Figure 2-4: OWLViz visualisation of the travel ontology

The figure shows that the sub-class relation is represented by the “is-a” relation. Moreover, it can be noticed that the graph lacks of properties, as the focus is only on the ontology’s class hierarchy.

The above tools are useful in visualising ontologies. Each one of them has their merits and demerits. Both of them are Protégé plugins that support the editor with variety of possible visualisation approaches. As these tools are developed for Java standalone applications, the use of the mentioned tools is not possible in our tool to visualise the resulting restricted views. This is due to our tool's platform, which is web-based. In Chapter 3, Section 3.3 VOWL is presented, which is used in the development of our tool.

2.5 Summary

In this chapter, general background about ontologies was presented. From the description logics background, we can establish that DLs are the backbone of knowledge representation, in which many ontology languages were introduced. The importance of OWL language, which was developed by the W3C, lies in providing machine processing language to build semantic web applications that exploit ontologies. Related approaches were discussed regarding the different segmentation methods and the available visualisation approaches of ontologies. In the next chapter, the background of the tools adapted in the development of the system is provided.

3 Used Tools

The background of the tools that formulate our system is illustrated in this chapter. **LETHE**, the core library of the system is presented in Section 3.1. OWL API, which is the main API that is needed to accomplish the system's functionalities, is shown in Section 3.2. Lastly, VOWL background representation is given in Section 3.3.

3.1 LETHE

As described previously, several of the applications including **PATO** and **Prompt** developed to produce restricted views of ontologies. Such modules are produced based on structure-based partitioning, which consider weak connections among the modules. Another method of producing modules is module extraction approach, which produce modules based on recursive traversal of a set of related elements. Thus, the resulting modules are ontologies that can be semantically weak, since they are produced based on their structural form. Uniform interpolation or forgetting is an important alternative method that aims to produce restricted views while preserving logical entailments by reducing the vocabulary of large ontologies [2]. The methods for producing a new ontology developed by Koopmann and Schmidt use the concept of a desired signature [2]. Depending on the desired signature, a new ontology may contain axioms not included in the input ontology. The signature can be class or property symbols that belong to a certain ontology. The result is a new ontology obtained from the original ontology by forgetting information expressed in terms of symbols that are not desired. The outcomes obtained by uniform interpolation differ slightly from the results given by forgetting. Uniform interpolation gives ontologies that contain information in the range of the desired symbols. On the other hand, results obtained by forgetting are concepts that are not in the range of the desired symbols. In other words, forgetting excludes concepts that are based on the desired symbols.

Uniform interpolation can also be used to compute logical differences between two ontologies [2], identifying those axioms shared between two ontologies over a certain signature. Logical differences can be used to identify whether two ontologies are S-inseparable, in other words, whether all of their entailments share the same S signature.

According to Koopmann [3], logical differences are computed as the uniform interpolant of a certain S . The reasoners can then establish whether the axioms of the second ontology are entailed by the first one. Those axioms that are not entailed by the first ontology represent new entailments over the selected signature. Logical differences can be used in applications that track changes in ontologies [3]. Another application of uniform interpolation is TBox abduction, which adds a set of axioms to an ontology in order to repair an ontology [2].

Among few implementations of uniform interpolation, **LETHE** is quite unique, which was developed by Patrick Koopmann [2]. Implementing uniform interpolation, forgetting, logical differences and TBox abduction, it can be used as a standalone tool or as a Java library. The standalone tool provides interface for uniform interpolation, which is the core functionality, using ALC, ALCH and SHQ. For ALCH DLs and SHQ DLs, **LETHE** forgets the symbols of TBoxes. In ALC DLs, the tool can also forget symbols involved in ABoxes axioms. To perform these tasks, it uses saturation-based reasoning methods, which help to preserve the semantics of ontologies by forgetting of symbols [3]. In **LETHE**, these methods produce the set of axioms not including any symbols that should be forgotten. For example, to forget symbol y , **LETHE** computes all of the entailments involving the symbol. The result of the process is an ontology whose axioms do not contain the symbol y [2].

3.2 OWL API

The OWL API is a high level programming interface supporting a number of functionalities: loading ontologies and creating, managing and saving them. Its general purpose reasoning functionality provides flexibility when using ontologies in other reasoner implementations, enabling the development of many different OWL editors and reasoners [27]. It also copes with the recent second version (OWL 2) [27].

The API supports processing of many types of OWL syntax, including RDF/XML, Turtle and OWL/XML. In addition, it can parse all the different syntaxes that it processes in an automatic mechanism without having to install a syntax-specific parser. Based on such

features, an editor can support conversion to different syntaxes [27]. The API also supports the use of different reasoners to detect axioms that are not entailed by the ontology.

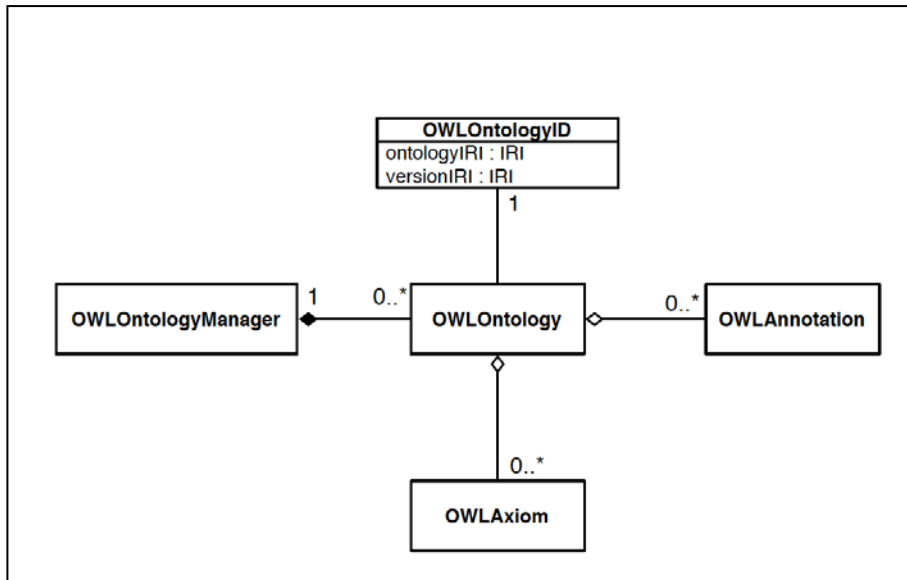


Figure 3-1: The UML class diagram of OWL API main interfaces [27]

The API design consists of a “set of interfaces”, providing functionalities for reasoning, creating and manipulating ontologies (Figure 3-1) [27]. An important aspect of the design is that it is based on the OWL Language structure, which represents the ontology in its simplest form, “as a set of axioms and annotations” [27]. Figure 3-1 shows the main interface of the design is the OWLOntology interface, which provides access to the ontology axioms through the OWLAxiom interface and to its annotations through the OWLAnnotation interface. In addition, each ontology instance is handled by its own manager through the OWLOntologyManager interface. This interface enables the management of OWL functionalities that include creating, loading, changing and saving ontologies. Moreover, entities that belong to a certain OWL ontology can be identified through the IRI, using the interface OWLOntologyID. The Internationalized Resource Identifiers (IRI) is used to identify ontologies and their elements, which is an address that is absolute (not relative) and refer to a certain ontology element [15]. For example, the address <http://www.co-ode.org/ontologies/pizza/pizza.owl#AsparagusTopping> refers to the AsparagusTopping class in the Pizza ontology [16].

For our tool, the two main functionalities provided by the API are loading and saving ontologies. The purpose of loading ontologies is to allow the user to choose the desired

ontology to work on in order to produce a restricted view. The purpose of the saving functionality is to preserve the resulting ontology in a certain directory after performing the **LETHE** process.

The API also supports the use of the Hermit reasoner used by **LETHE**, which provides powerful reasoning capabilities for ontologies based on description logics [28]. The API was chosen for its compatibility with the **LETHE** library; in fact, to ensure that the library works correctly, the API should be embedded with the library as a dependency during development stage.

3.3 VOWL

The Visual Notation for OWL Ontologies (VOWL) [29] is a language developed specifically to visualise ontologies constructed in the OWL language. It can visualise most OWL syntaxes, including RDF/XML, Turtle and OWL/XML. VOWL mainly enables visualisation of ontology TBoxes, including their classes, properties and datatypes; ABoxes can be optionally integrated using the visualisation canvas.

Additionally, VOWL offers the advantages of interaction with the graph and manipulation through force-directed methods [30], which enable dynamic interaction with the layout. Another advantage is supporting customisation by enabling options such as “datatype property” to be shown in the graph. Visual representation of ontologies is based on two types of VOWL building blocks: graphical primitives and colour scheme. VOWL visualises OWL elements using the graphical primitives and distinguishes between OWL and RDF elements through the use of different colours. This can be seen in Figure 3-2, where the `owl:Class` element is represented in light blue and the `rdfs:Class` element is represented in pink [29].



Figure 3-2: The difference between owl:Class and rdfs:Class representation [29]

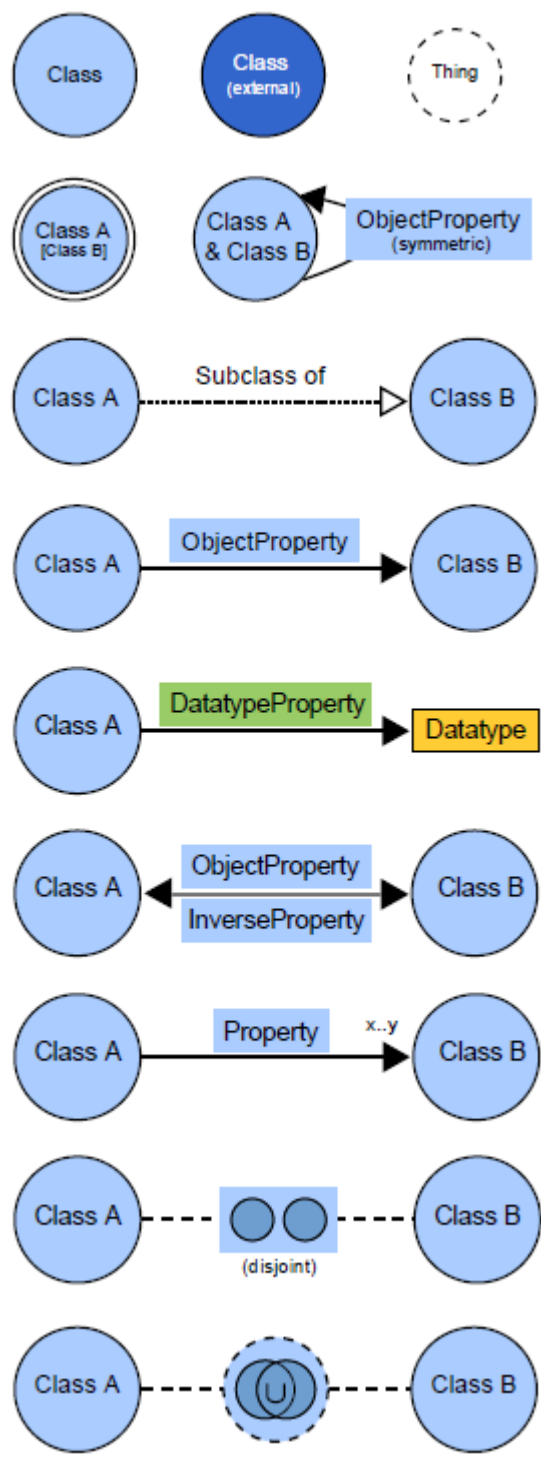


Figure 3-3: Illustration of VOWL visual elements [31]

VOWL representation of ontology elements vary in their colours and approaches. Figure 3-3 shows that some of the constructors used in ontologies are visualised by VOWL. For example, the disjunction relation is represented by the mathematical symbol \cup . This is the same for the conjunction relation, which is represented by \cap symbol. VOWL also represents external classes by dark blue colour and the hint “external” exists beneath the class name. External classes in VOWL are the classes that have IRIs that are different from that of the original ontology [31]. These IRIs could have been added to the JSON schema for the sake of visualisation, as this is the way VOWL visualises ontologies. The process of adding those IRIs could have been done because the ontology lacks of them. In the case of visualising cardinality restrictions and subclass relations, it follows a similar approach to UML class diagrams. This can be seen by (x .. y) numbers on top of the arrows for cardinality restrictions representation, and empty coloured triangle in the subclass relation. The remaining VOWL specifications of elements can be referred to in reference [32].

The VOWL graphical primitives are used to visualise classes, properties, datatypes and labels [29]. Figure 3-4 (Table (a)) shows that classes are represented by circles, and properties are represented by lines connecting the classes. Datatypes are illustrated by rectangles. Class circles vary in size within a given ontology; classes containing a larger number of individuals are visualised as larger circles. This rule does not apply to *Thing* class, which is the root class in every ontology. Although all of the individuals in an ontology belong to the root class *Thing*, it is represented as considerably smaller than other classes [29]. This size is fixed for all ontologies. Lines are used to connect classes, properties and datatypes, with arrowheads defining the domain classes and datatypes they belong to. If a class does not belong to a given domain, then the arrowheads points to `owl:Thing` class, except in the case of datatypes properties, where the arrowhead points to `rdfs:Literal`. This represents the general class for data values, in which every data value of an ontology is an instance of it. Rectangles are used to illustrate property labels identified by `rdfs:label`. If `rdfs:label` is empty, the property label can be identified from the last part of the IRI string [29].

VOWL colouring scheme specifications are based on elements' functionalities. Colour codes can be overridden during development to cope with desired application characteristics, but they should be compatible with VOWL specifications [29]. Figure 3-4 (Table (b)) shows the colour specification of elements, which clearly visualises the ontology's semantics, with general classes coloured lighter than external classes. Although the colouring of an ontology's elements is an important aspect of visualisation, this is not required for visualisation because the aforementioned graphical primitives enable elements to be distinguished by shape. Elements that should ideally be represented by colours for clarity can be visualised using text labels.

Table (a)







Primitive	Application
	classes
	properties
	property directions
	datatypes, property labels
	special classes and properties
	labels, cardinalities

Table (b)








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

Figure 3-4: Tables (a) and (b) illustrate VOWL graphical primitives and the colour schema respectively [29]

Two implementations were developed by the same VOWL developers to exploit the VOWL library, one of them is webVOWL, which is developed as a standalone application. The tool does not depend on a particular parser for the ontology parsing. Instead, it requires ontologies to be converted to JSON files that can be visualised using VOWL. The process of converting these ontologies is performed by using OWL2VOWL library, which converts OWL language to JSON schema [32]. Section 6.3 provides a comparison between the webVOWL tool and OntoGraf, which is one of the mentioned Protégé plugins that visualises ontologies.

3.4 Summary

This chapter provided comprehensive background about **LETHE**'s functionalities, OWL API and VOWL. These tools are used in the development of the system. The backbone of our system is **LETHE**, in which its functionalities are exploited to accomplish the project objectives. OWL API is necessary to be used in our system in order to exploit its functionalities and provide the dependencies required by **LETHE**. VOWL is an important component which provides visualisation means for the resulting ontologies. VOWL's background was illustrated to understand their different elements and specifications.

4 Design and Implementation

This chapter defines the system's requirements and its architecture in Section 4.1. It describes the platforms and methods used during the implementation of the project in Section 4.2. The main use cases' implementation conducted by the server are provided in Section 4.3. Lastly, the integration between the system's modules is described as well in Section 4.4.

4.1 Requirements and Overall System Architecture

The objectives of the project that are related to building the system's components helped in identifying the main requirements. These objectives are as follows:

- providing a visualisation features that help users in analysing ontologies
- developing a web application that exploit the **LETHE** library functionalities
- exploit the application in testing the **LETHE** library with different types of ontologies

The requirements are use cases that are performed by the client and the server. They are necessary to be identified in order to formulate the system's modules.

The use cases that the client performs are as follows:

- Uploading an input ontology to the system.
- Selecting from the services; Uniform Interpolation/Forgetting, Logical Differences and TBox Abduction.
 - If Uniform Interpolation/Forgetting service is selected, the client selects the desired symbols and the forgetting method.
 - If Logical Differences service is selected, the client inserts the second ontology, the approximation level and the forgetting method.
 - If TBox Abduction service is selected, the client selects TBox axioms, a set of signatures S and the forgetting method.
- Visualising the resulting ontology.
- Saving the resulting ontology to OWL format in local machine.
- Saving the resulting graph to SVG format in local machine.

The use cases that are conducted by the server are as follows:

- Loading the input ontology with its symbols.
- Processing of the ontology using **LETHE**, based on the selected services and symbols.
- Displaying the resulting ontology.
- Visualising the resulting ontology using webVOWL.
- Displaying the graph in the UI.

The main system components are the **LETHE** library, webVOWL, the ontology handler (OWL API) and the user interface (UI). Each of these components have a major role it plays to achieve the functionalities of the system. Figure 4-1 illustrates the components of the system architecture.

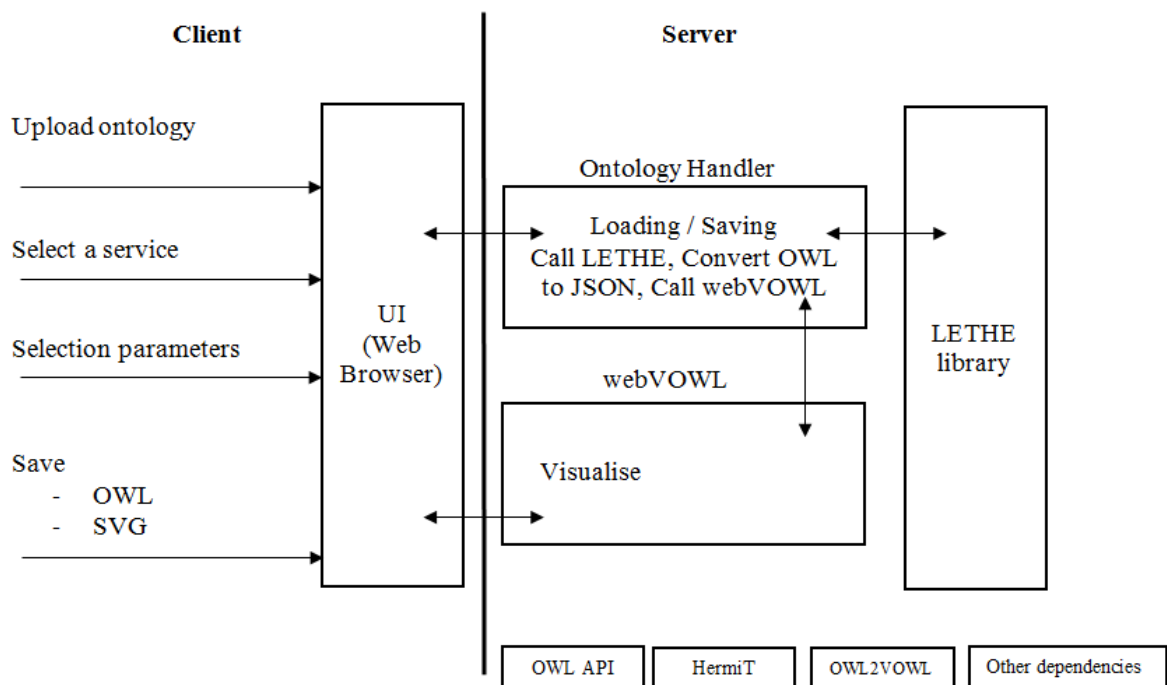


Figure 4-1: The system architecture represented as modules in the client and server sides

LETHE's role in the system architecture is to provide the services of uniform interpolation, logical differences and TBox abduction. The interpolation of the ontology depends on the type of the expressive language that is interpolated for. These expressive languages are ALCH, SHQ and ALC. Each of these languages represents the forgetting method that is used by the services (uniform interpolation and logical differences) to compute the resulting

ontology. As it was mentioned in the background section (3.1), the interpolation of TBoxes is conducted for ALCH and SHQ DLs. For ALC DLs, the interpolation of ABoxes is also performed.

Uniform interpolation/Forgetting service needs three basic parameters which are an input ontology, the forgetting method and a set of symbols in which the resulting views are based on. On the other hand, logical differences service needs four parameters which are two input ontologies to compute the logical differences between them, the forgetting method, approximation level and a set of symbols. The set of symbols is an optional parameter, as the user can also compute the differences based on common symbols that are shared between the two ontologies.

The other important component of our system is webVOWL. This tool represents the visualisation module of our system. webVOWL provides visualisation methods for the computed ontologies. In addition, it can be used by the user to visualise the uploaded ontology before computing them with a certain service. In order to use webVOWL for visualising ontologies provided by our system, OWL2VOWL library must be used. OWL2VOWL is a library that converts ontologies in OWL language to VOWL language in order to represent the elements of OWL graphically [31]. The VOWL language is represented by JSON string, which can be visualised by the use of D3 [54, 55]. D3 is a JavaScript library that uses HTML, CSS and SVG to visualise data. VOWL specification of OWL elements has been provided in details in Chapter 3, Section 3.3.

The ontology handler is a component that contributes to the system by handling ontologies functionalities: loading ontologies into the server, saving ontologies to the server's file system and **LETHE**'s dependencies on the OWL API, including the classes OWLEntity, OWLOntology and OWLLogicalAxiom.

The user interface (UI) plays an important role in the system architecture, as it is the gateway through which the system provides its services.

4.2 Implementation Platforms and Programming Language

The implementation platforms that were used during the development of the project are the following:

IntelliJ Integrated Development Environment (IDE)

The tool has been built using IntelliJ IDE which is a well-known integrated development environment used to develop java-language software applications [33]. It supports various technologies and frameworks that can assist in developing sophisticated applications. Moreover, it is compatible with most operating systems that support Java, including, Windows, Mac OS and Linux. The IDE is supported with several plugins that makes it easy to develop web applications. One of the web frameworks supported by it is Spring Web model-view-controller (MVC), which has been used in the development of our tool.

Versioning Control Systems - Git

An important aspect of developing a software application is to provide a back-up strategy of code and related materials. For this purpose, Git tool was used during the development of our tool. Git is a version control system created by Linus Torvalds, which supply developers with a mechanism to save code in a repository [34]. The repository is a database that saves versions of the code, so developers can revert back to the previous versions. This mechanism makes developers to have a strategy of backing up the code. Another feature of the tool is distributed development, meaning that more than one developer can contribute in the development of an application, with management methods provided by Git. These methods are related with the security of the distributed code, fixing code versions or overwriting them.

Git can be used by command line or by an interface. There are many Git clients that can be installed to use it in a graphical user interface. One of these clients is the popular, free GUI, GitHub application [35]. It can be used as a standalone application or through its website. The tool was developed in 2008, which uses the technologies provided by Git. It manages repositories and keeps logs of source code. Our tool uses GitHub to store the source code and keep versions of it. The repository can be accessed through the following link:

https://github.com/ghadaayash/LETHE_web

OWL API

Described previously in Chapter 3, Section 3.2.

LETHE Library

Described previously in Chapter 3, Section 3.1.

Java Application Web Framework - Spring MVC

In order to provide an efficient web services that cope with the provided **LETHE** Java library, Spring MVC [36, 37] model-view-controller has been used to build the system. Spring MVC follows an architectural pattern that deals with the system as having three different layers, which are presentation (UI), business (application logic) and database (resource management). In MVC architectural pattern, the model part represents the application logic layer. As MVC pattern does not represent the resource management layer, it is encapsulated by the model layer. The view part of the pattern represents the view layer (the presentation layer) of a system, which is the user interface. The user interface can be created using various view technologies including JSP, Velocity templates or XSLT views. The controller part of the pattern is responsible for handling events and requests conducted by the user in the view layer. The controller part of the pattern represents the business layer of systems architecture. The framework is request-driven, which uses a servlet that is responsible for sending all of the requests to controllers. This servlet is called a DispatcherServlet, which uses all of the features provided by the framework that makes it easy to create web-based applications.

Spring MVC framework features many advantages over other web frameworks, one outstanding feature is Inversion of Control (IoC) or Dependency injection, which states that objects do not rely on other objects in order to do a certain function [38]. Spring MVC is completely integrated with Dependency Injection (DI) feature. DI means that Spring framework uses assembler objects that assembles the different components of the system without depending on other objects to do the job that could increase coupling between the system components [38].

Another advantage of Spring MVC is the clear separation between the application layers that includes controllers, views and models [36]. This provides the ability to develop applications that is scalable and can be maintained in an independent way without making changes to the other application layers.

Other than the mentioned features, Spring MVC was chosen in the development of the tool because it provides comprehensive capabilities that helped in the achievement of the system's functionalities. One of these functionalities is using the MultipartFile interface provided by the framework to upload ontology files to the server. Figure 4-2 demonstrates the system's components in the perspective of Spring MVC request processing workflow. The figure shows the front controller (dispatcher servlet) which dispatches all of the requests made by the client to the controllers. In addition, we can see three different controllers, in which each one is responsible for a specific service that is provided by **LETHE**. The job of these controllers is to create the corresponding requested data (model) along with handling the requested URL that is triggered by the user. This data (model) can be in the form of the resulting ontology, after it was computed by the concerned controller. The template view represents the HTML code that is used to display the page.

The request processing workflow is as follows:

1. The user interact with the UI, by requesting a certain URL, such as localhost:8080/tl/logicaldifferences.html
2. The front controller (DispatcherServlet) delegates the request to the appropriate controller (uniform interpolation, logical differences, or TBox abduction controller).
3. The concerned controller sends back the requested data (model) to the front controller along with the suitable address of the template view, in this stage the controller can get help from other classes (such as the ontology handler classes). This stage represents the business layer of the application.
4. The front controller renders the response came by the controller to the concerned view template (at the client side).
5. The involved view returns the control to the front controller, meaning that it returns the HTML code mixed with the requested data.
6. The front controller sends the HTML code mixed with the requested data as a response to the user (the client browser). This stage represents the presentation layer of the application.
7. Finally, the browser displays the page.

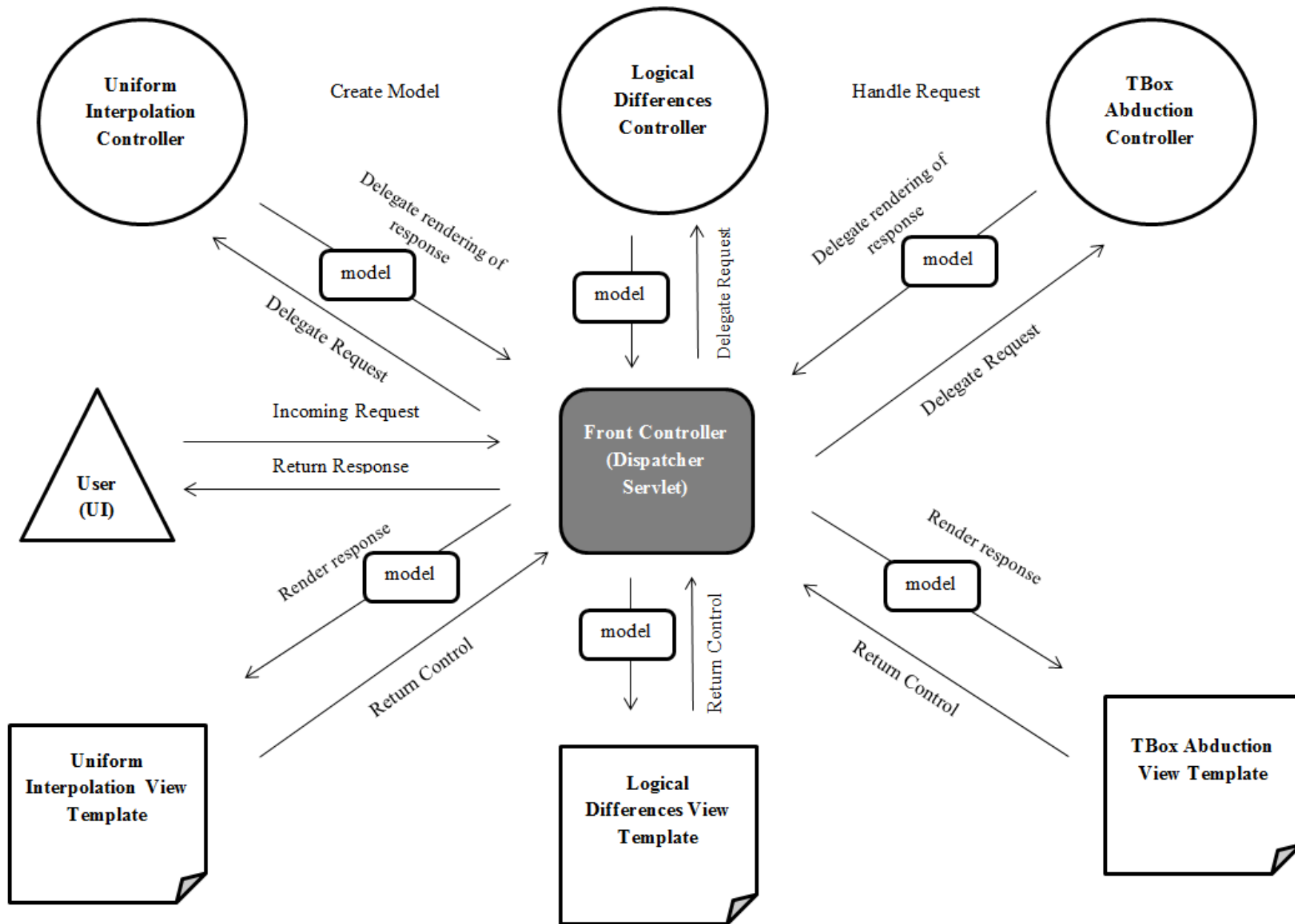


Figure 4-2: Illustration of the system's request processing workflow in Spring MVC

View Layer Framework - Apache Tiles

The presentation layer of the tool was developed using the Apache Tiles framework [60]. The framework was used for its provision of simplicity during the development of the user interface views. It provides view templates that allow the developer to easily manage the different user interface pages' components. These templates are comprised of fragments that pages are consists of. These fragments are then assembled during the runtime to form the overall page. To create these templates, they are defined in the Tiles configuration file. Each service in our system has its own template to make it easy to add its different view specifications, including the title of the service page, the body and the footer. Figure 4-3 illustrates part of the Tiles configuration file that contains the definitions for each system service. Taking the uniform interpolation page as an example, first, the template is created by defining the page that contains the header and the footer, which is called "firstTemplate". Then, the page body and title are defined by creating another page that extends the "firstTemplate", which is called "UniformInterpolation". As a result, we have three fragments that form the uniform interpolation page: the title, the body and the footer.

```
<!-- Uniform Interpolation Template -->
<definition name="firstTemplate" template="/WEB-INF/layout/classic-uniform.jsp">
  <put-attribute name="footer" value="/WEB-INF/layout/footer.jsp" />
</definition>

<definition name="UniformInterpolation" extends="firstTemplate">
  <put-attribute name="title" value="Uniform Interpolation" />
  <put-attribute name="body" value="/WEB-INF/jsp/UniformInterpolation.jsp" />
</definition>

<!-- Logical Differences Template -->
<definition name="secondTemplate" template="/WEB-INF/layout/classic-logical.jsp">
  <put-attribute name="footer" value="/WEB-INF/layout/footer.jsp" />
</definition>

<definition name="LogicalDifferences" extends="secondTemplate">
  <put-attribute name="title" value="Logical Differences" />
  <put-attribute name="body" value="/WEB-INF/jsp/LogicalDifferences.jsp" />
</definition>

<!-- TBox Abduction Template -->
<definition name="thirdTemplate" template="/WEB-INF/layout/classic-abduction.jsp">
  <put-attribute name="footer" value="/WEB-INF/layout/footer.jsp" />
</definition>

<definition name="TBoxAbduction" extends="thirdTemplate">
  <put-attribute name="title" value="TBox Abduction" />
  <put-attribute name="body" value="/WEB-INF/jsp/TBoxAbduction.jsp" />
</definition>
```

Figure 4-3: Code snippet of the Tiles configuration file

4.3 Use Cases' Implementation

This section illustrates the main use cases' implementation that is conducted by the server. The presentation of these functionalities is classified according to the package to which they belong. These packages include the controllers (uniform interpolation and logical differences controllers), the **LETHE** package and the ontology handler package.

- The controllers package

The controllers package contains classes that represent the services' controllers (uniform interpolation, logical differences and TBox abduction) that were described previously in Section 4.2. The following is a description of the main functionalities that each of the controllers contain.

- `handleRequest` functionality

Each of the controllers contains the method of handling view requests triggered by the user. For example, if the user requests the uniform interpolation page, it will be handled by the function and returns the suitable requested address. Figure 4-4 illustrates the code for the method that returns the view of the uniform interpolation page (`UniformInterpolation.jsp`).

```
public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception{
    return new ModelAndView("/WEB-INF/jsp/UniformInterpolation.jsp");
}
```

Figure 4-4: Code of `handleRequest` method in uniform interpolation controller class

- uploading ontologies functionality

This function is responsible for uploading ontologies to the file server. The method is called from the `ontologyFile` class, which is responsible for various ontology files functionalities, such as uploading ontologies or saving the generated JSON file to be visualised by `webVOWL`. Figure 4-5 shows the code for the `uploadFile` method exists in the `ontologyFile` class.

```

public OWLOntology uploadFile(MultipartFile file, OWLOntology ontology){
    try {
        String fileName = null;
        InputStream inputStream = null;
        OutputStream outputStream = null;
        if (file.getSize() > 0) {
            inputStream = file.getInputStream();
            System.out.println("size:" + file.getSize());
            fileName = "/Users/ghadahalghandi/Documents/LETHE_web/src/main/webapp/upload/"
                + file.getOriginalFilename();
            File convFile = new File(fileName);
            file.transferTo(convFile);
            OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
            ontology = manager.loadOntologyFromOntologyDocument(convFile.getAbsoluteFile());
            outputStream = new FileOutputStream(fileName);
            System.out.println("fileName:" + file.getOriginalFilename());
            ontologies.add(ontology);
            int readBytes = 0;
            byte[] buffer = file.getBytes();
            while ((readBytes = inputStream.read(buffer)) != -1) {
                outputStream.write(buffer, 0, readBytes);
            }
            outputStream.close();
            inputStream.close();
            //manager.removeOntology(ontology);
        }
        // .....
    } catch (Exception e) {
        e.printStackTrace();
    }
    return ontology;}

```

Figure 4-5: Code of uploadFile method in OntologyFile class

- populating uploaded ontology symbols

Both of the uniform interpolation and logical differences controllers contain the function of populating ontology symbols. These ontology symbols are of type OWLEntity. The function extracts ontology symbols that are of type object property and class from the set of OWLEntity symbols. This is done by calling the method (getSignature) on the uploaded ontology to get all of the ontology symbols. Then, an iteration over these symbols is conducted to obtain only the class and object property symbols. After that, these symbols are accumulated into a set of type OWLEntity symbols to be accepted by the uniform interpolation and logical differences functionalities in the **LETHE** package. Figure 4-6 illustrates the code snippet for this function.

```

Set<OWLEntity> owlEntitySet = ontology.getSignature();
Set<OWLEntity> newOwlEntitySet = new HashSet<>();
for (Iterator iterator = owlEntitySet.iterator(); iterator.hasNext(); )
{
    OWLEntity owlEntity = (OWLEntity) iterator.next();
    if (owlEntity.isOWLClass() || owlEntity.isOWLObjectProperty()){
        newOwlEntitySet.add(owlEntity);
    }
}

```

Figure 4-6: Code snippet for populating ontology symbols to the user

- converting computed ontologies to JSON file

The function of converting the computed ontologies to a JSON file to be visualised by webVOWL exists in each of the controllers. As webVOWL visualises ontologies by the existence of their IRIs, the IRI is first extracted from the computed ontologies that are obtained from uniform interpolation or logical differences. Then, the OWL2VOWL library will take the new ontology and its IRI as parameters and return the JSON string. This JSON string then is passed to a method that exists in the `OntologyFile` class that will return the JSON file. The code snippet in Figure 4-7 illustrates the process of converting an ontology to JSON using OWL2VOWL in order to visualise it in webVOWL.

```
IRI iri = manager.getOntologyDocumentIRI(newontology);
//System.out.println("\n_____ " + iri.toString());
Owl2Vowl owl2Vowl = new Owl2Vowl(newontology, iri.toString());
String s = owl2Vowl.getJsonAsString();
String curie = "test-ontology-iri";
System.out.println("_____ " + OntologyFile.SaveJsonFileForWebVOWL(curie, s));
String lastpart = ss.takeLastPartCurie(OntologyFile.SaveJsonFileForWebVOWL(curie, s));
session.setAttribute("curie", lastpart);
modelMap.addAttribute("jsonText", s);
```

Figure 4-7: Code snippet for converting an ontology to a JSON file

- passing JSON files to the view layer

For webVOWL to visualise the computed ontologies produced by our system, the name of the corresponding JSON file is passed to the view layer. This name is combined with the address of webVOWL's tool so that it can visualise the ontologies. Section 4.4 describes the integration mechanism between our tool and webVOWL. Figure 4-8 shows the code snippet for passing a JSON file name to the view layer. For example, if the user clicks on the visualise button to visualise a certain ontology, the name of its corresponding JSON file is sent to the uniform interpolation page (in the view layer). The page contains the address of webVOWL application which is <http://ghstestapp.local/#test-ontology-iri-4>. The "test-ontology-iri-4" is the name of the JSON file that will be visualised by webVOWL. Figure 4-9 illustrates the page of webVOWL including the address of a JSON file being visualised.

```
String JSONFileName = ss.takeLastPartCurie(OntologyFile.SaveJsonFileForWebVOWL(curie, s));
session.setAttribute("JSONFileName", JSONFileName);
```

Figure 4-8: Code snippet for passing a JSON file name to the view layer

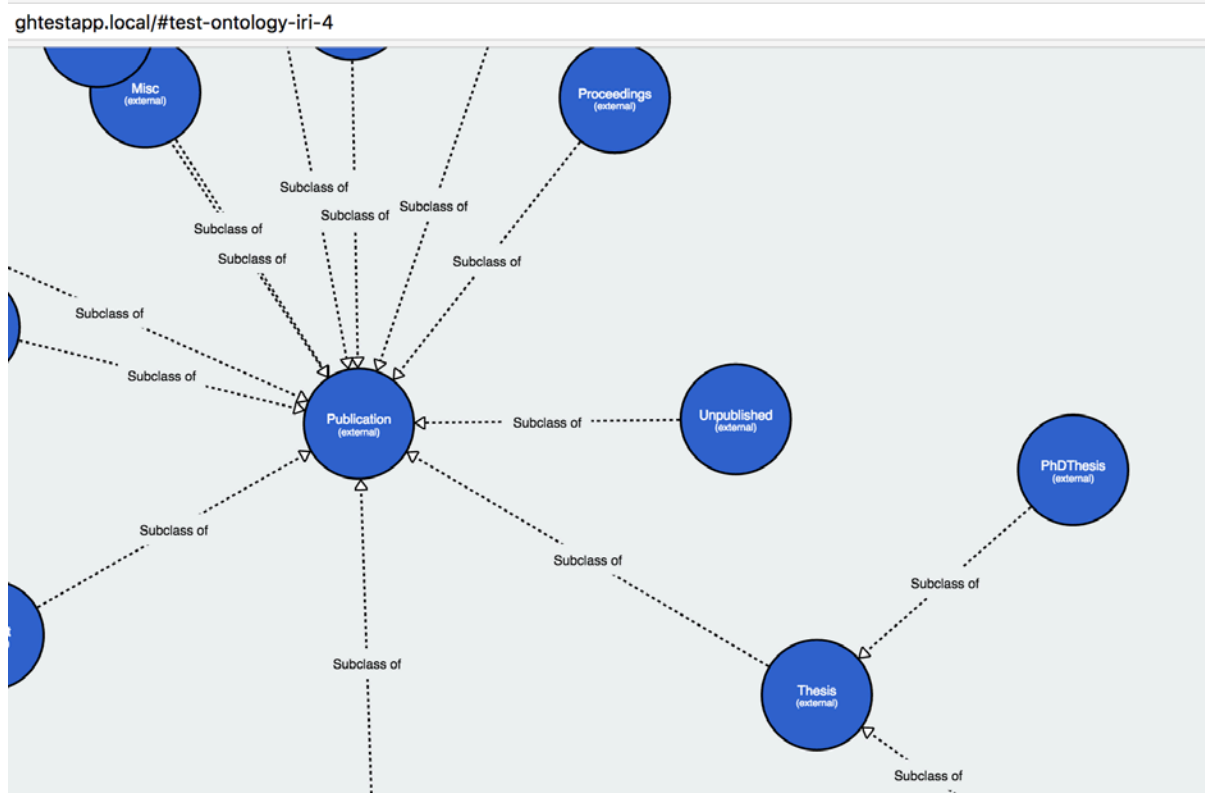


Figure 4-9: Illustration of webVOWL visualising the "test-ontology-iri-4" file

- **LETHE** package

The package contains the implementation for each of the **LETHE** functionalities: uniform interpolation and logical differences. Each function is represented by a class, as described below:

- Uniform interpolation class

The class contains function for each of the forgetting methods including ALCH TBoxes, SHQ TBoxes and ALC with ABoxes. The uniform interpolation for all of the forgetting methods accepts two basic parameters, which are the input ontology and a set of OWLEntity objects that represents the ontology symbols. These symbols are extracted from the uploaded ontology (input ontology). Figure 4-10 illustrates the uniform interpolation for ALCH

TBoxes method. The method returns a value of type `OWLOntology`, which is the computed ontology.

```
public OWLOntology alchInterpolation(OWLOntology inputOntology, Set<OWLEntity> symbols){  
    if (inputOntology.isEmpty() || symbols.isEmpty()){  
        System.out.println("Passed parameters are empty");  
        return null;  
    } else {  
        IOWLInterpolator interpolator = new AlchTBoxInterpolator();  
        OWLOntology interpolant = interpolator.uniformInterpolant(inputOntology, symbols);  
        return interpolant;  
    }  
}
```

Figure 4-10: Code for ALCH TBoxes interpolation in uniform interpolation class

- Logical differences class

The logical differences function is conducted similarly to the uniform interpolation function, as it is computed as the uniform interpolant of a certain set of symbols. This set of symbols can be common (shared between the two ontologies) or specified by the user. Each method of the class represents a certain forgetting method and the logical differences option, which can be based on common or specified symbols. Figure 4-11 illustrates the function that computes logical differences by interpolating for ALCH TBoxes. The function accepts three parameters: the first ontology, the second ontology and the approximation level. This function computes the logical differences based on common symbols between the two ontologies.

```
//Compute Logical Difference for common signature of the two ontologies:  
public Set<OWLLogicalAxiom> computeDiff_coms_alch(OWLOntology ontology1, OWLOntology ontology2, int approximationLevel)  
{  
    IOWLInterpolator interpolator = new AlchTBoxInterpolator();  
    LogicalDifferenceComputer ldc = new LogicalDifferenceComputer(interpolator);  
    Set<OWLLogicalAxiom> diff = ldc.logicalDifference(ontology1, ontology2, approximationLevel);  
    return diff;  
}
```

Figure 4-11: Code for the method that computes logical differences based on the ALCH TBoxes forgetting method (common symbols)

In cases in which the user would like to specify a set of symbols, another function will be performed, which is illustrated in Figure 4-12. This function is similar to the previous one. However, it takes an additional parameter, which is the set of symbols. The symbols (signature) are `OWLEntity` symbols. The method returns a set of axioms (`OWLLogicalAxiom`), which are then stored in an OWL file using a function exists in the ontology handler package.

```

// Compute Logical Difference for specified signature
public Set<OWLLogicalAxiom> computeDiff_s_alch(OWLOntology ontology1, OWLOntology ontology2, Set<OWLEntity> signature, int approximationLevel)
{
    //get interpolator based on?
    IOWLInterpolator interpolator = new AlchTBoxInterpolator();
    LogicalDifferenceComputer ldc = new LogicalDifferenceComputer(interpolator);
    Set<OWLLogicalAxiom> diff = ldc.logicalDifference(ontology1, ontology2, signature, approximationLevel);

    return diff;
}

```

Figure 4-12: Code for the method that computes the logical differences based on the ALCH TBoxes forgetting method (specified symbols)

- Ontology handler package

This package contains methods that are concerned with saving ontology files or reading them. Saving the computed ontologies in the file server is important for the user to be able to download them. Figure 4-13 shows the method that saves the computed ontologies by uniform interpolation. The saving is performed by the use of the OWLOntologyManger object that contains the method of saving ontologies to a certain directory. The method uses three parameters: the resulting ontology, the syntax of the ontology to be saved and a method that creates the IRI for the resulting ontology. The third parameter is important to convert the computed ontology to a JSON file that is visualised by webVOWL.

```

public File saveOntology(OWLOntology resultedOntology, String curie, String extension) throws IOException {
    OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
    String ontologyFilePath = getFileForOntology("/Users/gbadahalghamdi/Documents/LETHE_web/src/main/webapp/upload/", curie, extension);
    File newOntologyFile = new File(ontologyFilePath);
    newOntologyFile = newOntologyFile.getAbsolutePath();
    try {
        manager.saveOntology(resultedOntology, new RDFXMLOntologyFormat(), IRI.create(newOntologyFile.toURI()));
    } catch (OWLOntologyStorageException e) {
        e.printStackTrace();
    }

    return newOntologyFile;
}

```

Figure 4-13: Code for the method that saves ontologies computed by uniform interpolation

The function of saving OWL logical axioms that are obtained from the logical differences functionality is conducted by creating a new ontology file. The manager adds the set of the obtained axioms to the new ontology file. After that, the manager saves the new ontology to the file system directory. Figure 4-14 illustrates the code snippet for saving a set of OWL logical axioms in an OWL file.


```

public File saveAxioms(Set<OWLLogicalAxiom> resultedAxioms, String curie, String extension) throws IOException {
    OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
    String filePath = getFileForOntology("/Users/ghadahalghamdi/Documents/LETHE_web/src/main/webapp/upload/", curie, extension);
    File newOntologyFile = new File(filePath);
    //"/Users/ghadahalghamdi/Documents/LETHE_web/src/main/webapp/upload/newEntailments.owl";
    newOntologyFile=newOntologyFile.getAbsolutePath();
    try {
        OWLOntology ontology = manager.createOntology();
        for (Iterator<OWLLogicalAxiom> axiomIterator = resultedAxioms.iterator(); axiomIterator.hasNext(); ) {
            OWLAxiom axiom = axiomIterator.next();
            manager.addAxiom(ontology, axiom);
            manager.saveOntology(ontology, new RDFXMLOntologyFormat(), IRI.create(newOntologyFile.toURI()));
        }
    } catch ( OWLOntologyCreationException | OWLOntologyStorageException e) {
        e.printStackTrace();
    }
    return newOntologyFile;
}

```

Figure 4-14: Code for the method that saves ontologies computed by logical differences

4.4 System Integration

This section describes the integration mechanisms between **LETHE**, Spring MVC and webVOWL. It starts by briefly describing the integration between the **LETHE** library with Spring MVC framework. Then, a description of the backend of webVOWL is provided. In addition, an overview about the mechanism used in the integration between webVOWL and **LETHE_web** is presented.

The integration of **LETHE** within Spring MVC framework was straight forward as the **LETHE** Library is a Java based library. It was conducted by including the **LETHE** library into the project structure along with its dependencies. The **LETHE** library has many dependencies in order to work. Two important dependencies are the OWL API and the HermiT reasoner [28]. The reasoner was used for the role restriction resolution rule, specifically in the forgetting of roles in ALC and ALCH DLs [39]. This integration has accumulated in a web application that uses the services of the **LETHE** library. In order to distinguish the web application form other technologies used to build the system, it has been called **LETHE_web**.

webVOWL is an open source web application developed to provide an implementation of VOWL language for visualising ontologies. The application was developed using open web technologies on the client side. NodeJs must be used in order to install webVOWL for development purposes. NodeJs is a server that runs applications that uses JavaScript technologies [40]. NodeJs is suitable for running applications that mainly depends on open web standards, which acts as the backend of these applications.

In order to integrate webVOWL with **LETHE_web** for the visualisation of the resulting ontologies computed by the latter application, Nginx server was used, which is an HTTP server and a reverse proxy [41]. It has been used for the second feature provided by it which is reverse proxy. Reverse proxy means to obtain resources and information from another server. The proxy server then returns these resources from the other server to the client (the user) as if it was being generated from the proxy server itself [42]. Nginx server was suitable to be used for the purpose of linking between the tomcat server, which is the one used in **LETHE_web** system, and the NodeJs, which is the server used to run webVOWL. This methodology helped in completely using the resources provided by webVOWL in order to visualise the resulting ontologies.

Figure 4-15 shows a close illustration of the integration between tomcat server and webVOWL by using Nginx server in the middle of the two servers. The operation is conducted by writing specific Nginx configurations that permits the running of an HTTP web application, which is webVOWL in our case. After the writing of Nginx configurations, **LETHE_web** is configured to open the URL addresses of the HTTP application (webVOWL). The configuration of **LETHE_web** takes place by including (visualise) links that refers to the webVOWL application. For example, to visualise a certain ontology the user would normally click on the visualise button to visualise it in webVOWL. Thanks to Nginx server, webVOWL shows the graphical representation of the ontology as if it was integrated natively with our system.

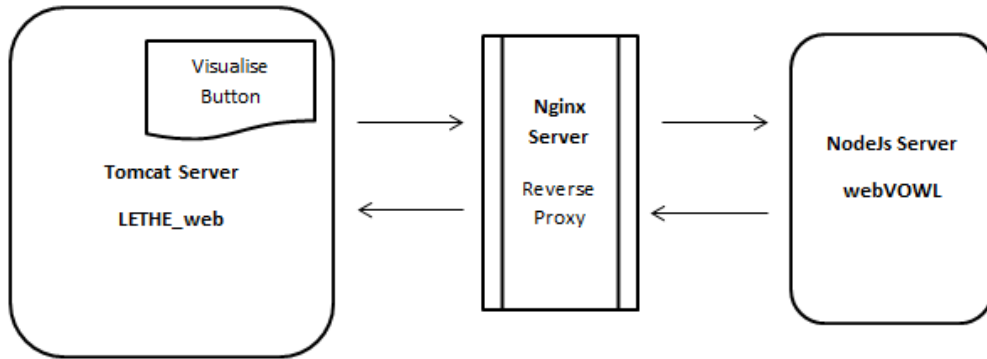


Figure 4-15: The system integration using Nginx server between Tomcat and NodeJs servers

4.5 Summary

The design and implementation processes began by identifying the functional requirements of the system, which includes defining the use cases conducted by the client and the server. The implementation platforms that were used during the development are necessary to be suitable with the application's requirements. Spring MVC is the backbone of the system's development process. The framework is request-driven that depends on a servlet which sends requests to the controllers. The process of our system's request workflow was identified in order to implement a system that functions accordingly. The main use cases implementation was described. Lastly, the integration mechanism between **LETHE_web** and webVOWL has been chosen according to the systems' servers requirements, which is described in Section 4.4.

5 System Testing and Illustration

In this chapter, an illustration of the system methods is provided. The illustration of using the system services are also presented in Section 5.2.

5.1 System Testing

System testing is a crucial phase of a software development. Through it we can be certain that the implemented system works as expected and functions in the right way. The purpose of the testing phase is to ensure that the system is free from faults. Therefore, our system was tested against two types of testing, which are unit testing and manual testing.

Unit Testing (White box testing)

The purpose of unit testing is to be certain that all of the methods behave according to the specified requirements. It is a type of white box testing, in which the developer tests the source code of an application line by line [43]. Unit testing is conducted by testing the smallest component of a portion of code, which is usually a method that accepts input parameters and return an output (return value). This kind of testing is suitable for a test driven development (TDD) methodology. In our tool, JUnit framework was used, which is a well-known testing framework that is used to test code written in Java language. It can be used for testing many types of components including web components [44].

The purpose of unit testing is to make sure that each function of the system works correctly. The tests were conducted against the following system packages

- Controllers
- LETHE
- Ontology Handler

The controllers package was tested against the handling of HTTP requests and responses, meaning the correctness of showing the intended view or page that corresponds to a certain controller. For example, the uniform interpolation controller should show its corresponding page (view) which is `UniformInterpolation.html`.

LETHE package included the functionalities of uniform interpolation, forgetting and logical differences. Each of them represents a class, which have many functions that correspond to a forgetting method. The testing involved each function of every class. The ontology handler package has many functionalities including uploading, saving and reading ontologies. Table 1 shows the category of tests, purpose, the number of tests and its outcome. Figure 5-1 illustrates the running of tests after it was accumulated into a test suite.

Category	Purpose	Number of tests	Outcome
Controllers package	To check the correctness of handling HTTP requests.	8	Pass
LETHE package	To check the correctness of processing LETHE functionalities in our tool.	12	Pass
Ontology Handler	To test the different functionalities of handling ontologies including uploading and saving them.	11	Pass

*Table 1: Summary of the unit tests performed on **LETHE_web***



*Figure 5-1: Illustration of tests under running in IntelliJ performed on **LETHE_web***

Moreover, print statements were included in order to check that the output results are accomplished in the intended way. Table 2 shows the categories of manual tests conducted as well as its outcomes.

Category	Purpose	Number of tests	Outcome
Uploading/ saving ontology files	To be certain the correct ontology was uploaded or saved to the file system	2	Pass
Populating ontology symbols/ selection of ontology symbols	To be certain the correct set of symbols were populated based on the uploaded ontology as well as the correct selection of symbols	2	Pass
The input parameters	To be certain the correct parameters were inserted into the system	3	Pass

Table 2: Summary of the print statements tests performed on LETHE_web

Manual tests (Black box testing)

Manual testing is the process of testing the system after it is built. It involves the developer or the tester to test the application as an end user. It is a type of black box testing, meaning that the source code of the system being under test is not shown, as if the system is a black box [43]. Many test cases have been considered while manually testing the system including the variety of input parameters such as ontologies, symbols, forgetting methods, modes, the easiness of dealing with the user interface and the correctness of the results. The results of performing manual tests helped in identifying the systems defects and clarified areas that can be improved.

5.2 LETHE Web-Analyser: System Illustration

This section illustrates the use cases of the system that the user performs. These use cases include using uniform interpolation and logical differences services. The section starts by presenting the uniform interpolation interface, the logical differences interface and webVOWL interface.

5.2.1 Uniform Interpolation Interface

The index page is shown in Figure 5-2. It contains links for each of the system's functionalities which are uniform interpolation, logical differences and TBox abduction. The implementation of TBox abduction is partially conducted. Therefore, the focus of this description is regarding uniform interpolation and logical differences. It can be seen that the interface is simply designed, with clear headings and fonts.

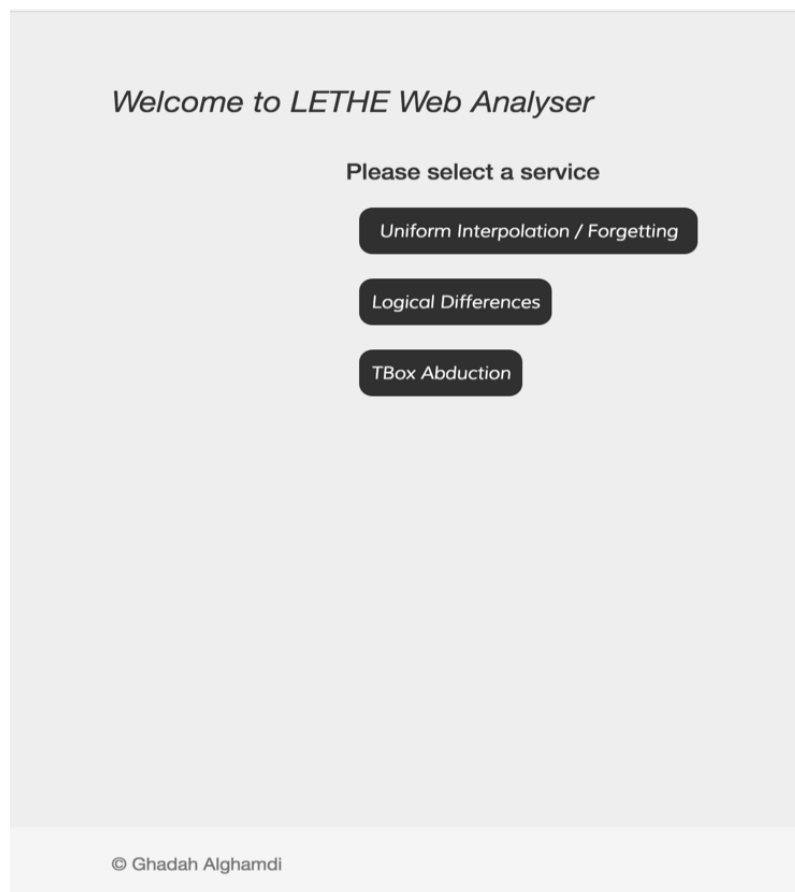


Figure 5-2: Main Page of LETHE Web Analyser

Figure 5-3 illustrates the overall uniform interpolation page. In the page, the steps of applying the function are clearly shown with numbers. This provides simplicity and clarity to the user when using the tool.

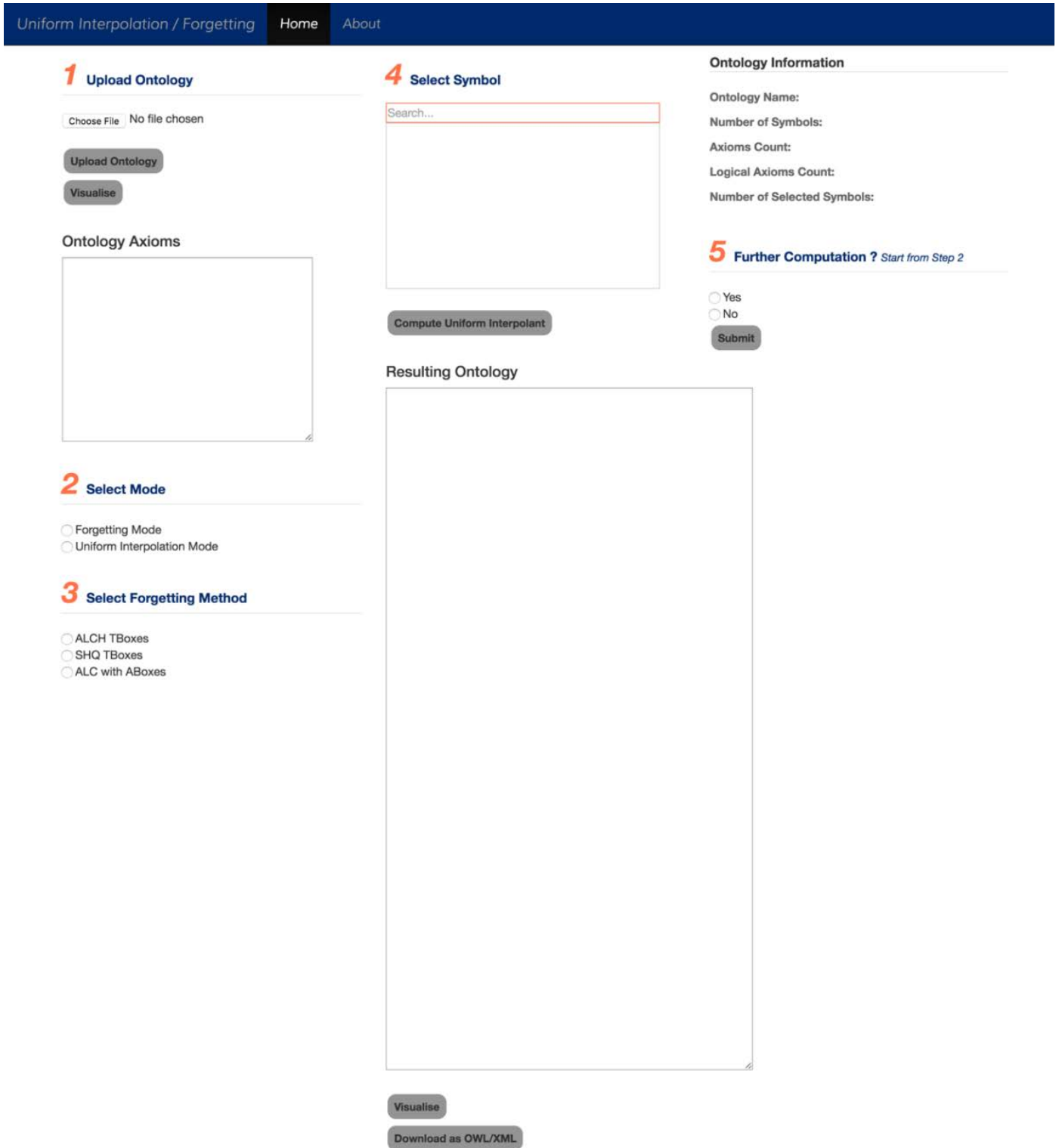


Figure 5-3: The overall page of uniform interpolation interface

© Ghadah Alghamdi

To describe each use case of the uniform interpolation / forgetting page, an example of using a simple ontology is given. The ontology was developed by Nick Knouf for the purposes of describing BibTeX tool terms [45]. BibTeX is a well-known tool used to list references in a certain format. The ontology provides small set of classes that describes the entries used in BibTeX tool [46, 47]. Some of the main terms are **Book**, **Manual**, **MasterThesis** and **Proceedings**.

The first step is uploading the ontology (Figure 5-4). After uploading the ontology, the axioms of the ontology can be browsed from the ontology axioms box. Also, the user can press on **Visualise** button in order to visualise the ontology in webVOWL. webVOWL interface is presented in Section 5.2.3.

1 Upload Ontology

Choose File bibtex.owl

Upload Ontology

Visualise

Ontology Axioms

```
Article ⊑ Entry
Book ⊑ Entry
Booklet ⊑ Entry
Conference ≡ Inproceedings
Conference ⊑ Entry
Inbook ⊑ Entry
Incollection ⊑ Entry
Inproceedings ⊑ Entry
Manual ⊑ Entry
Mastersthesis ⊑ Entry
```

Figure 5-4: First Step (upload ontology) in uniform interpolation page

The second step is selecting the mode of the computation, which can be either forgetting or uniform interpolation (Figure 5-5). The differences between the two modes is in the inclusion of symbols, meaning that if the forgetting mode is chosen, then the purpose of choosing symbols is to eliminate axioms that contains those symbols. While in uniform interpolation mode, the resulting ontology axioms are limited to the chosen symbols. The third step is choosing the forgetting method (ALCH TBoxes, SHQ TBoxes or ALC with ABox) (Figure 5-5).

The image shows a web interface for uniform interpolation. It is divided into three sections:

- 2 Select Mode:** Contains two radio buttons: "Forgetting Mode" (unselected) and "Uniform Interpolation Mode" (selected).
- 3 Select Forgetting Method:** Contains three radio buttons: "ALCH TBoxes" (selected), "SHQ TBoxes" (unselected), and "ALC with ABoxes" (unselected).
- 4 Select Symbol:** Contains a search input field with the placeholder text "Search...". Below the search field is a list of symbols: Article, Book, Booklet, Conference, Entry, Inbook, Incollection, Inproceedings, Manual, and Mastersthesis. At the bottom of this section is a button labeled "Compute Uniform Interpolant".

Figure 5-5: Steps 2 (select mode) and 3 (select forgetting method) in uniform interpolation page

Figure 5-6: Step 4 (select symbol) in uniform interpolation page

The fourth step is selecting the symbols in order to perform the computation based on them. The select symbol box is accompanied with a search box, in which the user can start typing a certain symbol and the symbols will be filtered according to the typed characters. Figures 5-6 and 5-7 illustrate the concept. After the selection of symbols takes place, the user can click on “Compute Uniform Interpolant” button in order to start the computation of producing a restricted view ontology. Figure 5-8 illustrates the set of chosen symbols and the resulting ontology.

4 Select Symbol

Book
Booklet
Inbook

Figure 5-5: Symbols filtration in Step 4 in uniform interpolation page

4 Select Symbol

Article
Book
Booklet
Conference
Entry
Inbook
Incollection
Inproceedings
Manual
Mastersthesis
...

Resulting Ontology

- Article Entry
- Book Entry
- Booklet Entry
- Conference Entry
- Inbook Entry
- Incollection Entry

Figure 5-6: Resulting ontology after the selection of symbols in uniform interpolation page

Moreover, the resulting ontology can be downloaded as OWL/XML format and visualised using webVOWL. Figure 5-9 shows the buttons for the aforementioned functions, which are located under the resulting ontology box.

Figure 5-7: (Visualise) and (Download as OWL/XML) buttons located under resulting ontology box in uniform interpolation page

The last optional step is to perform further computation of uniform interpolation / forgetting functionality (Figure 5-10). This step is applied to the last ontology computed. Thus, the symbols are extracted from the last resulting view, and the select symbols box is refreshed with the symbols occurring in that view. Figure 5-10 illustrates the uploaded ontology information, which is the ontology name, the number of symbols that the ontology contains, the number of axioms, the number of logical axioms and the number of selected symbols. The ontology name is the name of a .owl file. The number of symbols is the number of entities that are of type OWLClass or OWLObjectProperty that the ontology contains. The number of axioms is the axioms count of the ontology, while the number of logical axioms is the number of axioms that are neither declaration axioms nor annotation axioms [48]. Lastly, the number of selected symbols is the number of the symbols that were selected by the user.

Ontology Information

Ontology Name: **bibtex.owl**

Number of Symbols: **15**

Axioms Count: **322**

Logical Axioms Count: **139**

Number of Selected Symbols: **8**

5 Further Computation ? *Start from Step 2*

Yes

No

Submit

Figure 5-8: Ontology information section and fifth step (further computation) in uniform interpolation page

5.2.2 Logical Differences Page

The second main page of our web browser is the logical differences page. There are six steps required by the user in order to use the function. The first and second steps involve uploading the first and second ontology, which are shown in the steps (1 and 2) of Figure 5-11. After the uploading of ontologies, the user can press on “Visualise first ontology” or “Visualise second ontology” buttons in order to use webVOWL for the visualisation of the first and second ontologies. The third step is to choose the logical differences mode of computation, which can be either based on common symbols that the two ontologies share, or on specific symbols (Step 3 in Figure 5-11). If the user selects the specific symbols option, then the logical differences of the two ontologies are computed based on the second ontology’s symbols. The fourth step is to select the forgetting method that is used during the computation of logical differences (Step 4 in Figure 5-11). The fifth step is selecting symbols, which is illustrated by Step 5 in Figure 5-11. This step depends on the third step, if the user selected common symbols option, then this step is skipped.

The last step is to specify the approximation level, which is specified by a given number (Step 6 in Figure 5-11); this number determines the level of approximation of logical differences in case the uploaded ontologies are cyclic. In addition, the information of the uploaded ontologies is displayed. Afterwards, the user can press the “Compute logical differences” button to start the process of computing the logical differences between the two ontologies. The ontology obtained can be downloaded as an OWL/XML file or it can be visualised using webVOWL. Concrete example of logical differences is given in a case study presented in Chapter 6, Section 6.2.

1 Upload first ontology

Choose File No file chosen

2 Upload second ontology

Choose File No file chosen

Upload Ontologies

Visualise First Ontology

Visualise Second Ontology

3 Compute logical differences based on

- Common Symbols
- Specified Symbols

Submit

4 Select Forgetting Method

- ALCH TBoxes
- SHQ TBoxes
- ALC with ABoxes

5 Select Symbol

Number of Selected Symbols:

6 Approximation Level

Compute Logical Differences

First Ontology Information

First Ontology Name:

Number of Symbols:

Axioms Count:

Logical Axioms Count:

Second Ontology Information

Second Ontology Name:

Number of Symbols:

Axioms Count:

Logical Axioms Count:

Resulting Axioms

Visualise

Download as OWL/XML

Figure 5-9: The overall page of logical differences interface

5.2.3 webVOWL Interface

The use cases of webVOWL interface are mainly to browse the ontologies and interact with the nodes and edges of the graph of the ontology. Figure 5-13 illustrates the overall page of webVOWL. It shows the graph of the Bibtex ontology that we have used in the illustration of uniform interpolation use cases. The interface is divided to two main sections. The first section holds the graphical representation of the ontology, and the second section represents the ontology information. Moreover, there is a thin tool bar in the bottom of the page that holds controllers including Export, Gravity, Filter, Modes, Reset and Pause. The most important controller is the export menu, in which the user can export the graph as JSON or SVG files. In the second section of the interface we have five subsections that include the ontology title, description, metadata, statistics and selection details.

The title subsection encloses more detailed information such as the version number of the ontology, the authors and the language in which it was written. The description sub-section presents short description of the ontology. The metadata represents the information that is extracted from annotation axioms in the ontology (Figure 5-12). The existence of the information of these sections depends on the uploaded ontology specifications.

The statistics subsection shows statistics about the ontology's classes, object properties, data properties, individuals, nodes and edges numbers of the graph. The selection details subsection clarifies information about the selected node in the graph.

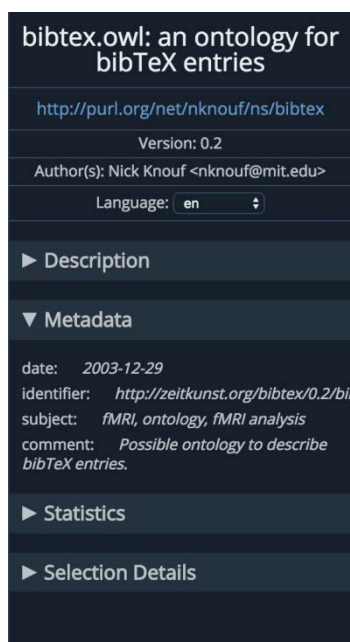


Figure 5-10: Metadata subsection of webVOWL interface

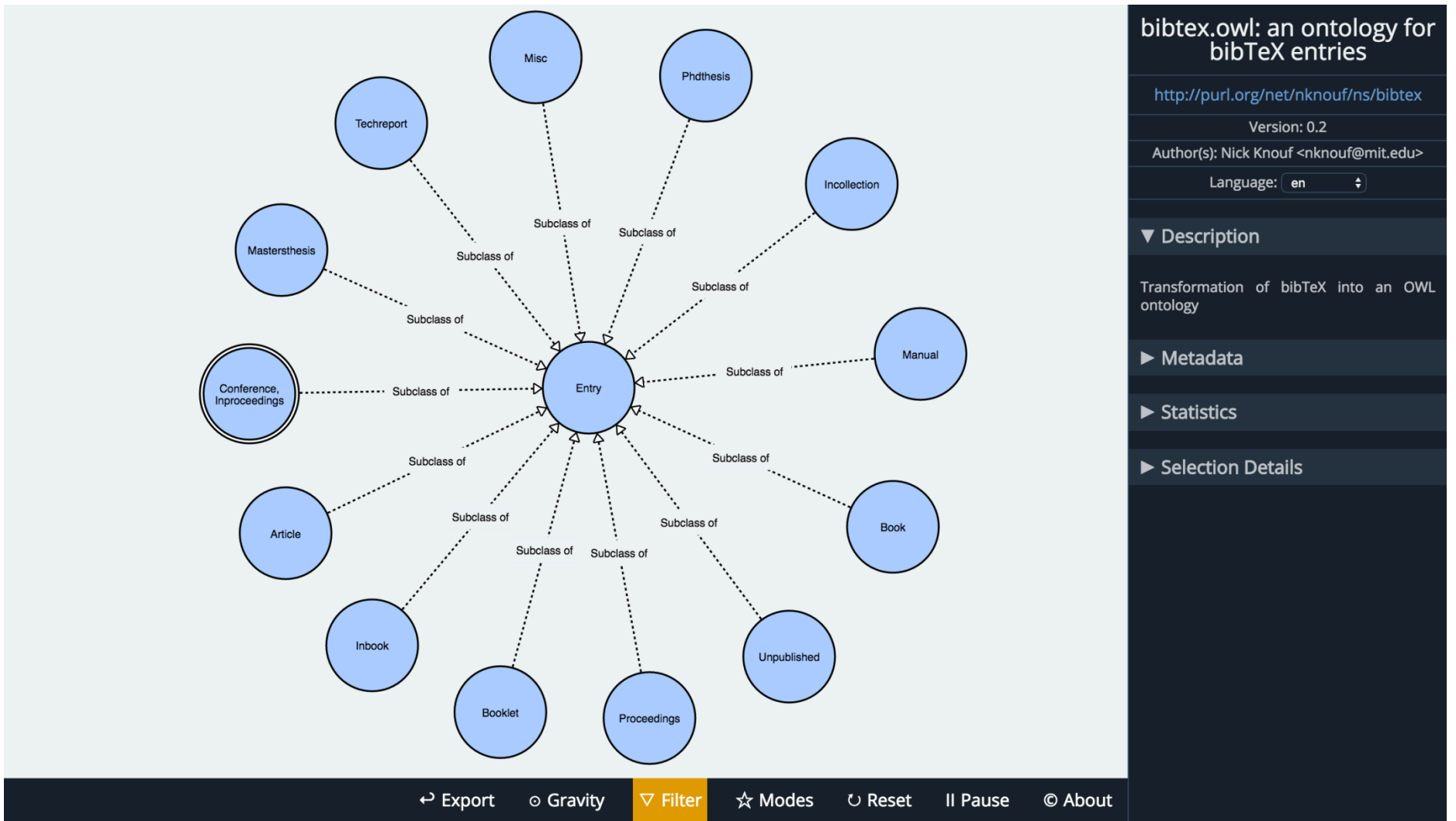


Figure 5-11: The overall page of webVOWL interface showing the visualisation of bibtex ontology

5.3 Summary

This chapter presented the testing methodologies used to test the system. These methods include unit testing and manual testing. They both ensure efficiency and validation of the system to function properly. The description of the system's use cases with regard to its services was provided.

6 Case Studies

This chapter presents case studies of uniform interpolation and logical differences functionalities. The purpose of the case studies is to evaluate the aforementioned functionalities and to illustrate their usefulness. The case studies also involved analysing of the resulting ontologies. In addition, a comparison between VOWL Library and visualisation library OntoGraf, provided by Protégé, is presented in Section 6.3. Moreover, the chapter presents a comparison of our system with mentioned modularisation tool (**PATO**) in Section 6.4

6.1 Uniform Interpolation and Forgetting Case Study

The following case studies were conducted in order to illustrate the evaluation results of each of the system's functionalities. Each of the case studies (uniform interpolation / forgetting and logical differences) consists of the following elements:

- The specified functionality of the system.
- The ontology background.
- The parameters inserted to evaluate the function.
- The resulting ontology based on the input parameters.
- Discussion of the results considering the following:
 - Whether the result corresponds with **LETHE**'s outcomes.
 - Whether the result can be clearly illustrated using webVOWL.
 - The usefulness of the result to the user.
- Conclusions

This case study is focused on evaluating the core functionality of the system which is uniform interpolation. This function aims to produce a restricted view of an ontology with the preservation of all of its logical entailments. The ontology that has been chosen in order to clearly illustrate the outcomes is the travel ontology.

The travel ontology was developed by Holger Knublauch [49]. It contains common terms that relate to travelling. The travel ontology contained five main concepts, which are accommodation, accommodation rating, activity, contact and destination. These concepts are

classes that have subclasses. The existence of subclasses makes us have more specified and clarified concepts. For example, the *Activity* class consists of four subclasses which are *Adventure*, *Relaxation*, *Sightseeing* and *Sports*. The activity class describes the type of activities that a person might perform during a trip or a holiday. The ontology also describes the concepts of the type of destination that someone might seek. Some of these destinations are: *BackpackersDestination*, *Beach*, *BudgetHotelDestination* and *FamilyDestination*. All of the aforementioned destinations are subclasses of the class *Destination*. In addition, the ontology includes six object properties, which are **hasAccommodation**, **hasActivity**, **hasContact**, **hasRating** and **isOfferedAt**. The ontology was created with the well-known editor, Protégé for tutorial purposes. The expressivity of it is ALC extended with role hierarchies, transitive roles, nominals, inverse properties and cardinality restrictions (SHOIN).

In order to apply our system to the travel ontology, we consider a restricted view of the ontology based on the destinations symbols. Suppose the user in this case selects the following symbols:

- the *Destination* class, and
- the sub-classes:
 - *BackpackersDestination*
 - *Beach*
 - *BudgetHotelDestination*
 - *FamilyDestination*
 - *QuietDestination*
 - *RetireeDestination*
 - *RuralArea*
 - *Farmland* sub-class of *RuralArea*
 - *NationalPark* sub-class of *RuralArea*
 - *UrbanArea*
 - *City* sub-class of *UrbanArea*
 - *Capital* sub-class of *City*
 - *Town* sub-lclass of *UrbanArea*

The chosen mode was uniform interpolation, meaning that the resulting ontology contains axioms that are in the scope of the chosen symbols. The forgetting method that was used was the one for ALCH TBoxes.

After the input parameters were specified, which are the travel ontology, the above symbols, the uniform interpolation mode and the forgetting method (ALCH TBoxes), the resulting ontology is produced. Figure 6-1 shows the computed ontology in a readable format. Figure 6-3 shows the webVOWL visualisation of the restricted view ontology, which is based on the destinations symbols. A number of classes can be seen in the graph, including *Town* and *City*, which are subclasses of *UrbanArea*, *NationalPark* is a subclass of *RuralArea*, and *UrbanArea* and *RuralArea* are subclasses of *Destination* class.

Figure 6-1 shows that the computed ontology is excluded from axioms that involve object properties, which are **hasAccommodation**, **hasActivity**, **hasContact**, **hasPart**, **hasRating** and **isOfferedAt**. Some of these object properties are not related to the chosen symbols that are based on destination concepts. However, the **hasPart** property relates to the *Destination* class, which can be seen in the original ontology axioms (Figure 6-2). These axioms are $\exists \text{hasPart.T} \sqsubseteq \text{Destination}$, meaning that a **hasPart** property exists in *Thing* class which is a subclass of *Destination* class and the axiom $\text{T} \sqsubseteq \forall \text{hasPart.Destination}$ which means that all of the **hasPart** property values are in the *Destination* class which the *Thing* class is part of. The reason of not including properties in the resulting ontology is because ALCH TBoxes forgetting method supports the forgetting of roles (object properties) [2] and object properties were not chosen to be included in the restricted view.

Resulted Ontology

```
(QuietDestination ⊓ FamilyDestination) ⊑ ⊥
(RuralArea ⊓ UrbanArea) ⊑ ⊥
BackpackersDestination ⊑ Destination
Beach ⊑ Destination
BudgetHotelDestination ⊑ Destination
Capital ⊑ City
City ⊑ UrbanArea
Destination ⊑ (QuietDestination ⊔ FamilyDestination)
Farmland ⊑ RuralArea
NationalPark ⊑ RuralArea
QuietDestination ⊑ Destination
RuralArea ⊑ Destination
Town ⊑ UrbanArea
UrbanArea ⊑ Destination
```

Ontology Axioms

```
∃hasActivity.T ⊑ Destination
∃hasContact.T ⊑ Activity
∃hasPart.T ⊑ Destination
∃hasRating.T ⊑ Accommodation
∃isOfferedAt.T ⊑ Activity
T ⊑ ∀hasAccommodation.Accommodation
T ⊑ ∀hasActivity.Activity
T ⊑ ∀hasContact.Contact
T ⊑ ∀hasPart.Destination
T ⊑ ∀hasRating.AccommodationRating
T ⊑ ∀isOfferedAt.Destination
```

Figure 6-2: Some of the travel original ontology axioms

Figure 6-1: The resulting ontology based on ALCH TBoxes forgetting method (uniform interpolation mode)

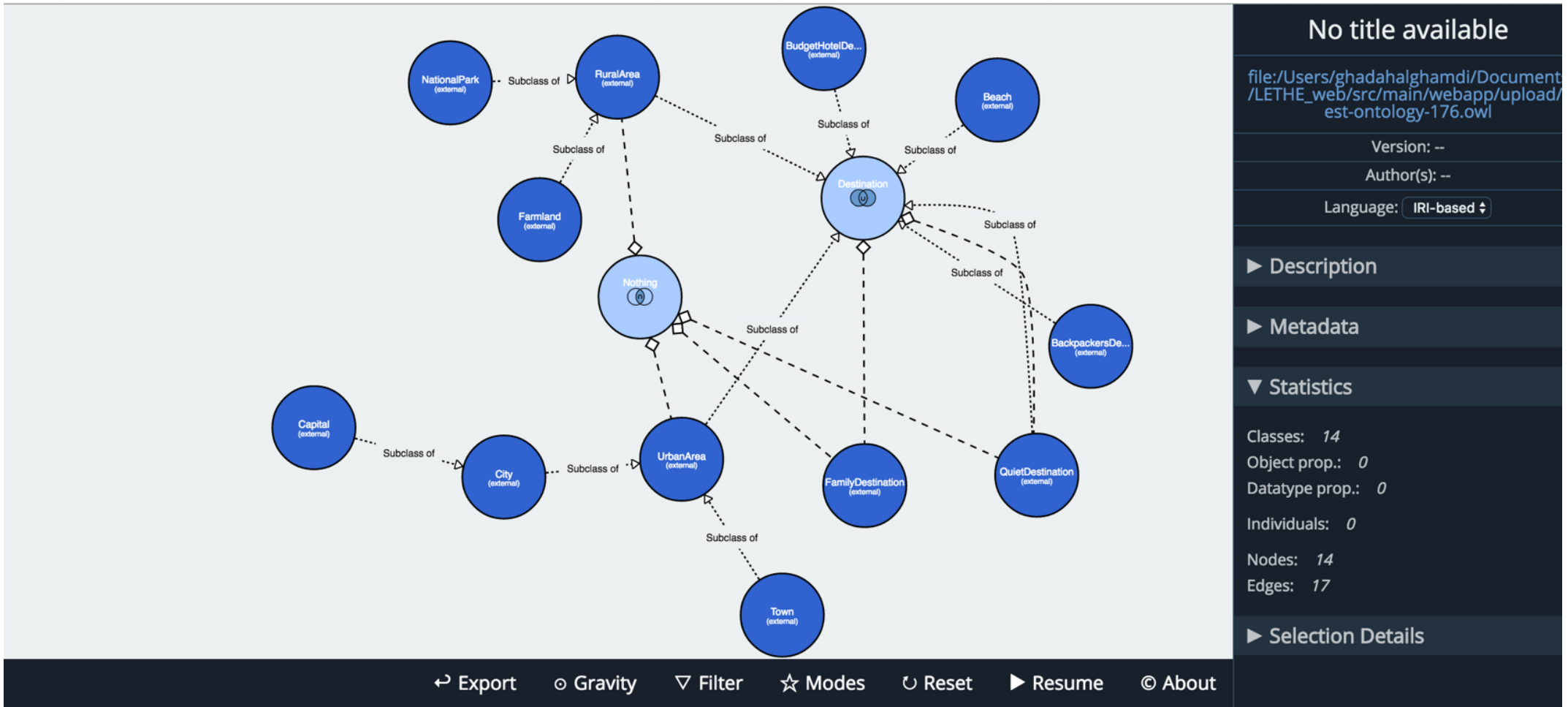


Figure 6-3: webVOWL visualisation of the resulting ontology based on ALCH TBoxes forgetting method (uniform interpolation mode)

In case the object properties that have relationships with destination terms were included, then the results will include the chosen properties. Figure 6-4 illustrates the resulting ontology after including the properties: **hasAccommodation**, **hasActivity**, **hasPart** and **isOfferedAt**.

Resulting Ontology

```

(QuietDestination ⊓ FamilyDestination) ⊆ ⊥
(RuralArea ⊓ UrbanArea) ⊆ ⊥
BackpackersDestination ⊆ Destination
Beach ⊆ Destination
BudgetHotelDestination ⊆ Destination
Capital ⊆ City
City ⊆ UrbanArea
Destination ⊆ (QuietDestination ⊔ FamilyDestination)
Farmland ⊆ RuralArea
NationalPark ⊆ RuralArea
QuietDestination ⊆ Destination
RuralArea ⊆ Destination
Town ⊆ UrbanArea
UrbanArea ⊆ Destination
∃hasAccommodation.⊤ ⊆ Destination
∃hasActivity.⊤ ⊆ Destination
∃hasPart.⊤ ⊆ Destination
⊤ ⊆ ∀hasPart.Destination
⊤ ⊆ ∀isOfferedAt.Destination

```

Figure 6-4: The resulting ontology based on ALCH TBoxes forgetting method after including object properties (uniform interpolation mode)

If we use the forgetting method for SHQ TBoxes, then the resulting ontology contains some object properties that the ontology has. As SHQ TBoxes forgetting method does not support the forgetting of object properties, there was no attempt made by **LETHE** to eliminate roles. The computed ontology is shown in Figure 6-5 in readable format. The figure shows the axiom

$$(\mathbf{BackpackersDestination} \sqcap \mathbf{Capital}) \sqsubseteq \geq 2\mathbf{hasActivity}.\top$$

This axiom means that the result of the disjointness between *BackpackersDestination* and *Capital* is part of the restriction on the property **hasActivity** that has two minimum cardinalities in the *Thing* class. In another words, the result of the intersection of *BackpackersDestination* and *Capital* classes implies that there at least two activities in the *Thing* class. Another axiom from the resulting ontology is

$$\mathbf{FamilyDestination} \sqsubseteq (\geq 2\mathbf{hasActivity}.\top \sqcap \mathbf{Destination} \sqcap \exists\mathbf{hasAccommodation}.\top)$$

This axiom means that a *FamilyDestination* should have at least two activities and an accommodation.

Resulting Ontology

```

(BackpackersDestination ⊓ Capital) ⊑ ≥2hasActivity.⊤
(BudgetHotelDestination ⊓ Destination ⊓ NationalPark) ⊑ BackpackersDestination
(BudgetHotelDestination ⊓ NationalPark) ⊑ ≥2hasAccommodation.⊤
(City ⊓ NationalPark) ⊑ ≥2hasAccommodation.⊤
(Destination ⊓ ∃hasAccommodation.⊤) ⊑ (FamilyDestination ⊔ ≤1hasActivity.⊤)
(NationalPark ⊓ Capital) ⊑ ≥2hasActivity.⊤
(RuralArea ⊓ UrbanArea) ⊑ ⊥
BackpackersDestination ⊑ Destination
BackpackersDestination ⊑ ∃hasAccommodation.⊤
BackpackersDestination ⊑ ∃hasActivity.⊤
Beach ⊑ Destination
BudgetHotelDestination ⊑ Destination
BudgetHotelDestination ⊑ ∃hasAccommodation.⊤
Capital ⊑ City
Capital ⊑ ∃hasActivity.⊤
City ⊑ UrbanArea
City ⊑ ∃hasAccommodation.⊤
Destination ⊑ (QuietDestination ⊔ FamilyDestination)
FamilyDestination ⊑ (≥2hasActivity.⊤ ⊓ Destination ⊓ ∃hasAccommodation.⊤)
Farmland ⊑ RuralArea
NationalPark ⊑ RuralArea
NationalPark ⊑ ∃hasAccommodation.⊤
NationalPark ⊑ ∃hasActivity.⊤
QuietDestination ⊑ (Destination ⊓ ¬FamilyDestination)
RuralArea ⊑ Destination
Town ⊑ UrbanArea
UrbanArea ⊑ Destination
trans(hasPart)
∃hasAccommodation.⊤ ⊑ Destination
∃hasActivity.⊤ ⊑ Destination
∃hasPart.⊤ ⊑ Destination
⊤ ⊑ ∀hasPart.Destination
⊤ ⊑ visOfferedAt.Destination

```

Figure 6-5: The resulting ontology based on SHQ TBoxes forgetting method (uniform interpolation mode)

The following figure (Figure 6-6) illustrates the graph of the resulting ontology in webVOWL after applying SHQ TBoxes forgetting method. The figure shows that the statistics shown by webVOWL include four object properties, which are **hasAccommodation**, **hasActivity**, **isOfferedBy** and the transitive property **hasPart**. However, the computed view did not include **hasRating** and **hasContact** properties that the original ontology has, as both of them do not relate to the *Destination* class, and this was conducted as a consequent of forgetting concepts that do not involve the *Destination* symbols. This can be seen in the axioms from the original ontology:

$$\begin{aligned}
\top &\sqsubseteq \forall \text{hasRating. AccommodationRating} \\
&\exists \text{hasRating. } \top \sqsubseteq \text{Accommodation} \\
&\exists \text{hasContact. } \top \sqsubseteq \text{Activity} \\
\top &\sqsubseteq \forall \text{hasContact. Contact}
\end{aligned}$$

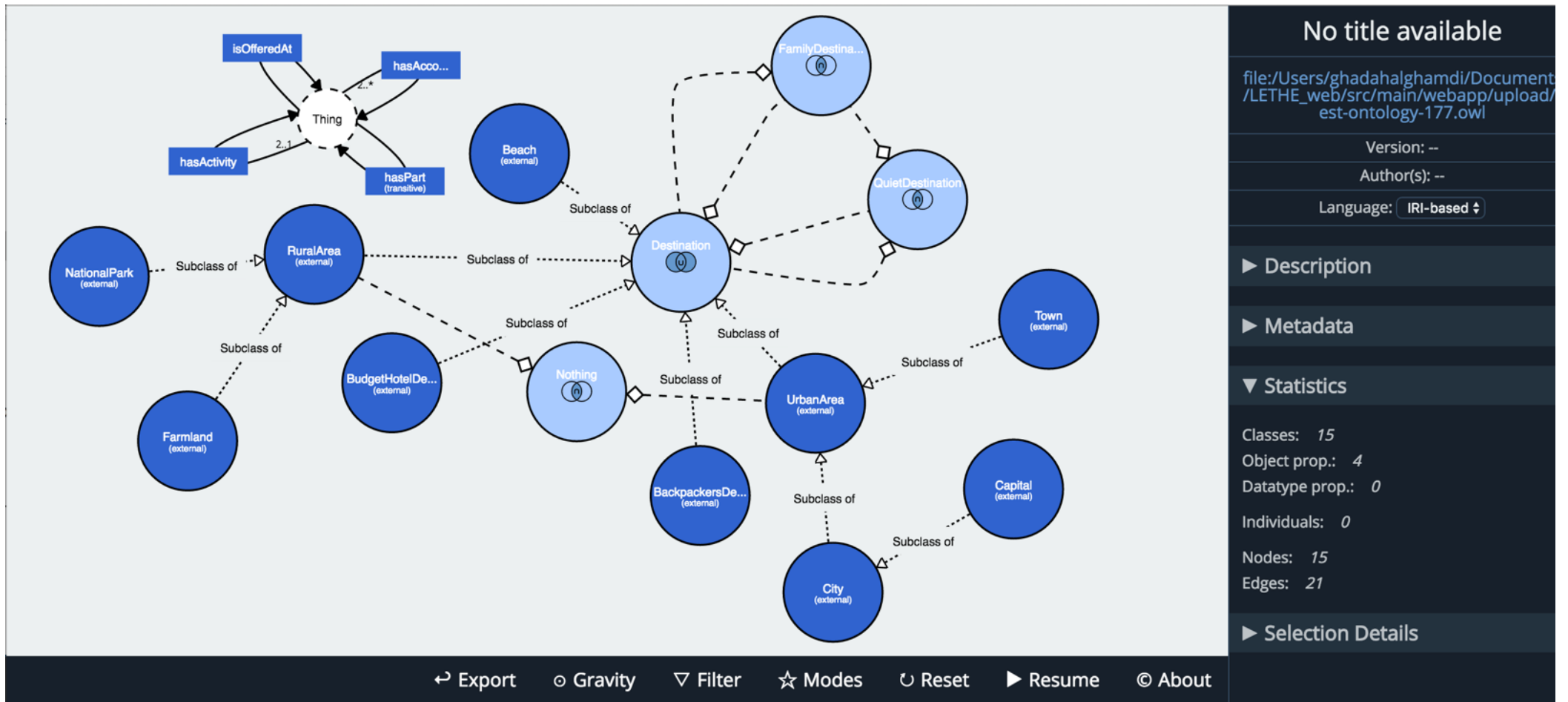


Figure 6-6: webVOWL visualisation of the resulting ontology based on SHQ TBoxes forgetting method (uniform interpolation mode)

Selecting the forgetting method for ALC with ABoxes resulted in a restricted view ontology that contains individuals, as this forgetting method also supports ABox axioms in the ontology. Figure 6-7 illustrates the axioms of the ontology in readable format. Some of the axioms including **Beach(BondiBeach)** and **Beach(CurrawongBeach)** implies that *BondiBeach* and *CurrawongBeach* are members of the class **Beach** which is a sub-class of the **Destination** class. The resulting view excluded properties as this forgetting method supports forgetting of them. The subsequent figure illustrates the graph of the resulting ontology in webVOWL after applying ALC with ABoxes forgetting method (Figure 6-8). The statistics section of webVOWL interface shows the number of individuals included in the ontology is 13. Also, the individuals in the visualisation of webVOWL are represented by numbers. For example, the class **Destination** has three members (individuals). The number is seen in the middle of the class **Destination** node in the graph.

Resulting Ontology

(BudgetHotelDestination \sqcap Destination \sqcap BackpackersDestination) \sqsubseteq (BudgetHotelDestination \sqcup BackpackersDestination)
(BudgetHotelDestination \sqcap Destination \sqcap BackpackersDestination) \sqsubseteq BackpackersDestination
(BudgetHotelDestination \sqcap Destination \sqcap NationalPark) \sqsubseteq (BudgetHotelDestination \sqcup BackpackersDestination)
(BudgetHotelDestination \sqcap Destination \sqcap NationalPark) \sqsubseteq BackpackersDestination
(BudgetHotelDestination \sqcap Destination) \sqsubseteq BudgetHotelDestination
(Destination \sqcap BackpackersDestination) \sqsubseteq (BudgetHotelDestination \sqcup BackpackersDestination)
(Destination \sqcap BackpackersDestination) \sqsubseteq BackpackersDestination
(Destination \sqcap NationalPark \sqcap BackpackersDestination) \sqsubseteq (BudgetHotelDestination \sqcup BackpackersDestination)
(Destination \sqcap NationalPark \sqcap BackpackersDestination) \sqsubseteq BackpackersDestination
(RuralArea \sqcap UrbanArea) \sqsubseteq \perp
BackpackersDestination \sqsubseteq Destination
Beach \sqsubseteq Destination
Beach(BondiBeach)
Beach(CurrawongBeach)
BudgetHotelDestination \sqsubseteq Destination
Capital \sqsubseteq City
Capital \sqsubseteq Destination
Capital(Canberra)
Capital(Sydney)
City \sqsubseteq Destination
City \sqsubseteq UrbanArea
City(Cairns)
Destination \sqsubseteq (QuietDestination \sqcup FamilyDestination)
Destination(BondiBeach)
Destination(CurrawongBeach)
Destination(Sydney)
Farmland \sqsubseteq RuralArea
NationalPark \sqsubseteq Destination
NationalPark \sqsubseteq RuralArea
NationalPark(BlueMountains)
NationalPark(Warrumbungles)
QuietDestination \sqsubseteq (Destination \sqcap \neg FamilyDestination)
RuralArea \sqsubseteq Destination
RuralArea(CapeYork)
RuralArea(Woomera)
Town \sqsubseteq UrbanArea
Town(Coonabarabran)
UrbanArea \sqsubseteq Destination
\top (FourSeasons)
\top (OneStarRating)
\top (ThreeStarRating)

Figure 6-7: The resulting ontology based on ALC with ABoxes forgetting method (uniform interpolation mode)

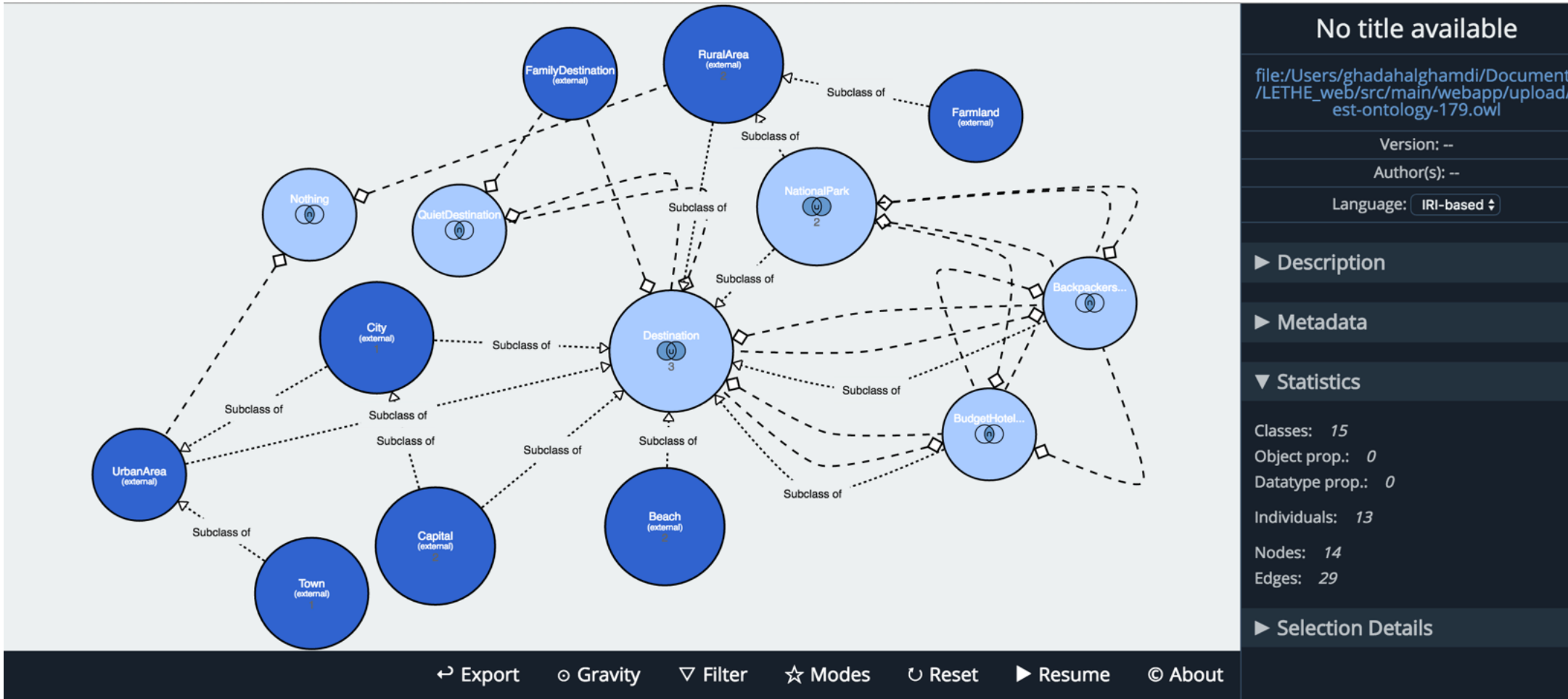


Figure 6-8: webVOWL visualisation of the resulting ontology based on ALC with ABoxes forgetting method (uniform interpolation mode)

The following table summarises the results of applying the three forgetting methods on the travel ontology using uniform interpolation mode (Table 3). The axioms of the results were divided to two categories, similar and noticeably different axioms that distinguish the results of a certain forgetting method from the other.

Forgetting Method	Similar resulting axioms among all forgetting methods	Resulting noticeably different axioms
ALCH TBoxes	<ul style="list-style-type: none"> - $(\text{RuralArea} \sqcap \text{UrbanArea}) \sqsubseteq \perp$ - $\text{BackpackersDestination} \sqsubseteq \text{Destination}$ - $\text{Beach} \sqsubseteq \text{Destination}$ - $\text{BudgetHotelDestination} \sqsubseteq \text{Destination}$ - $\text{Capital} \sqsubseteq \text{City}$ - $\text{City} \sqsubseteq \text{UrbanArea}$ - $\text{Destination} \sqsubseteq (\text{QuietDestination} \sqcup \text{FamilyDestination})$ - $\text{Farmland} \sqsubseteq \text{RuralArea}$ - $\text{NationalPark} \sqsubseteq \text{RuralArea}$ - $\text{RuralArea} \sqsubseteq \text{Destination}$ - $\text{Town} \sqsubseteq \text{UrbanArea}$ - $\text{UrbanArea} \sqsubseteq \text{Destination}$ 	<ul style="list-style-type: none"> - $(\text{QuietDestination} \sqcap \text{FamilyDestination}) \sqsubseteq \perp$ - $\text{QuietDestination} \sqsubseteq \text{Destination}$
SHQ TBoxes		<ul style="list-style-type: none"> - $(\text{BackpackersDestination} \sqcap \text{Capital}) \sqsubseteq \geq 2 \text{hasActivity.T}$ - $(\text{BudgetHotelDestination} \sqcap \text{NationalPark}) \sqsubseteq \geq 2 \text{hasAccommodation.T}$ - $(\text{City} \sqcap \text{NationalPark}) \sqsubseteq \geq 2 \text{hasAccommodation.T}$ - $\text{trans}(\text{hasPart})$
ALC with ABoxes	<ul style="list-style-type: none"> - $\text{Farmland} \sqsubseteq \text{RuralArea}$ - $\text{NationalPark} \sqsubseteq \text{RuralArea}$ - $\text{RuralArea} \sqsubseteq \text{Destination}$ - $\text{Town} \sqsubseteq \text{UrbanArea}$ - $\text{UrbanArea} \sqsubseteq \text{Destination}$ 	<ul style="list-style-type: none"> - $\text{Beach}(\text{BondiBeach})$ - $\text{Beach}(\text{CurrawongBeach})$ - $\text{Town}(\text{Coonabarabran})$

Table 3: Summary of the resulting axioms of all forgetting methods (uniform interpolation mode)

The table shows that the restricted view computed using ALCH TBoxes method includes two axioms not obtained with the other methods. The first axiom says the intersection of *QuietDestination* class and *FamilyDestination* class is part of the *Nothing* class. In other words, the two classes are complements to each other (each one negates the other) since they are part (sub-classes) of the *Nothing* (\perp) class. This axiom clearly illustrates a TBox axiom. In addition, in case the user does not specify object properties to be included in the computed ontology, then the method excludes all of the object properties of the original ontology from the resulting view. This is the case for ALC and ALCH DLs.

Moreover, it is noticeable that the information obtained by SHQ TBoxes forgetting method is more detailed and precise. This is because SHQ is the most expressive language among ALCH and ALC, which illustrates that **LETHE** exploits this additional expressivity to express and capture more for this case than the other two cases. For example, the axiom

$$(\text{City} \sqcap \text{NationalPark}) \sqsubseteq \geq 2 \text{hasAccommodation.T}$$

means that there are at least two accommodations in the result of intersecting *City* and *NationalPark* classes. Finally, the results of the ALC with ABoxes mode generally include ABox axioms. This is shown in the axiom **Town(Coonabarabran)**, which means that *Coonabarabran* is a member of the class *Town*.

The computed uniform interpolants exclude *RetireeDestination*, which is a subclass of *Destination* while using all of the three forgetting methods. This is because *RetireeDestination* includes an individual *ThreeStarRating* in its equivalent axiom. The *ThreeStarRating* in this case represents a nominal constructor. **LETHE** Library excludes axioms that contain nominals as this case does not cope with **LETHE's** functionality of producing uniform interpolants or when forgetting axioms (in the forgetting mode).

The second mode is forgetting, in which the axioms that represent concepts based on the selected symbols are eliminated from the ontology. Thus, the resulting ontology is a restricted view ontology that contains axioms outside the scope of the chosen symbols. In addition to the destination symbols that we applied in uniform interpolation mode, concepts based on accommodation symbols were chosen as well. The *Accommodation* symbols are as follows:

- the *Accommodation* class, and
- the sub-classes:
 - o *BedAndBreakfast*
 - o *BudgetAccommodation*
 - o *Campground*
 - o *Hotel*
 - *LuxuryHotel* subclass of *Hotel*
- the *AccommodationRating* class, and
- the **hasAccommodation** object property

The information in the resulting ontology was related to *Activity* and *Contact* symbols which represent description of the activities that can be done during a trip, including adventure activities such as Bunjee jumping or safari, relaxation activities including sunbathing and yoga activities, and sightseeing activities. These concepts represent the remaining symbols of the ontology (that were not chosen by the user).

First, the ALCH TBoxes forgetting method was chosen, in which the resulting ontology included object properties (**hasActivity**, **hasContact**, **isOfferedAt**), and the class *Activity* and its subclasses including *Adventure*, *Sports*, and the *Contact* class. The inclusion of these object properties is due to their relation to the *Activity* and *Contact* classes. However, it has forgotten the rest of the object properties (**hasAccommodation**, **hasPart** and **hasRating**) as they are involved in the *Accommodation* and *Destination* symbols, which are symbols that were chosen to be forgotten. In addition, forgetting of object properties took place because of the aforementioned reason of its support of object properties forgetting. The resulting ontology and its graph are shown in Figures 6-9 and 6-10, respectively.

Resulting Ontology

(Adventure \sqcap Relaxation) $\sqsubseteq \perp$
(Adventure \sqcap Sightseeing) $\sqsubseteq \perp$
(Adventure \sqcap Sports) $\sqsubseteq \perp$
(Relaxation \sqcap Sightseeing) $\sqsubseteq \perp$
(Relaxation \sqcap Sports) $\sqsubseteq \perp$
(Sightseeing \sqcap Sports) $\sqsubseteq \perp$
Adventure \sqsubseteq Activity
BunjeeJumping \sqsubseteq Adventure
Hiking \sqsubseteq Sports
Museums \sqsubseteq Sightseeing
Relaxation \sqsubseteq Activity
Safari \sqsubseteq Adventure
Safari \sqsubseteq Sightseeing
Sightseeing \sqsubseteq Activity
Sports \sqsubseteq Activity
Sunbathing \sqsubseteq Relaxation
Surfing \sqsubseteq Sports
Yoga \sqsubseteq Relaxation
\exists hasContact. \top \sqsubseteq Activity
\exists isOfferedAt. \top \sqsubseteq Activity
\top \sqsubseteq \forall hasActivity.Activity
\top \sqsubseteq \forall hasContact.Contact

Figure 6-9: The resulting ontology based on ALCH TBoxes forgetting method (forgetting mode)

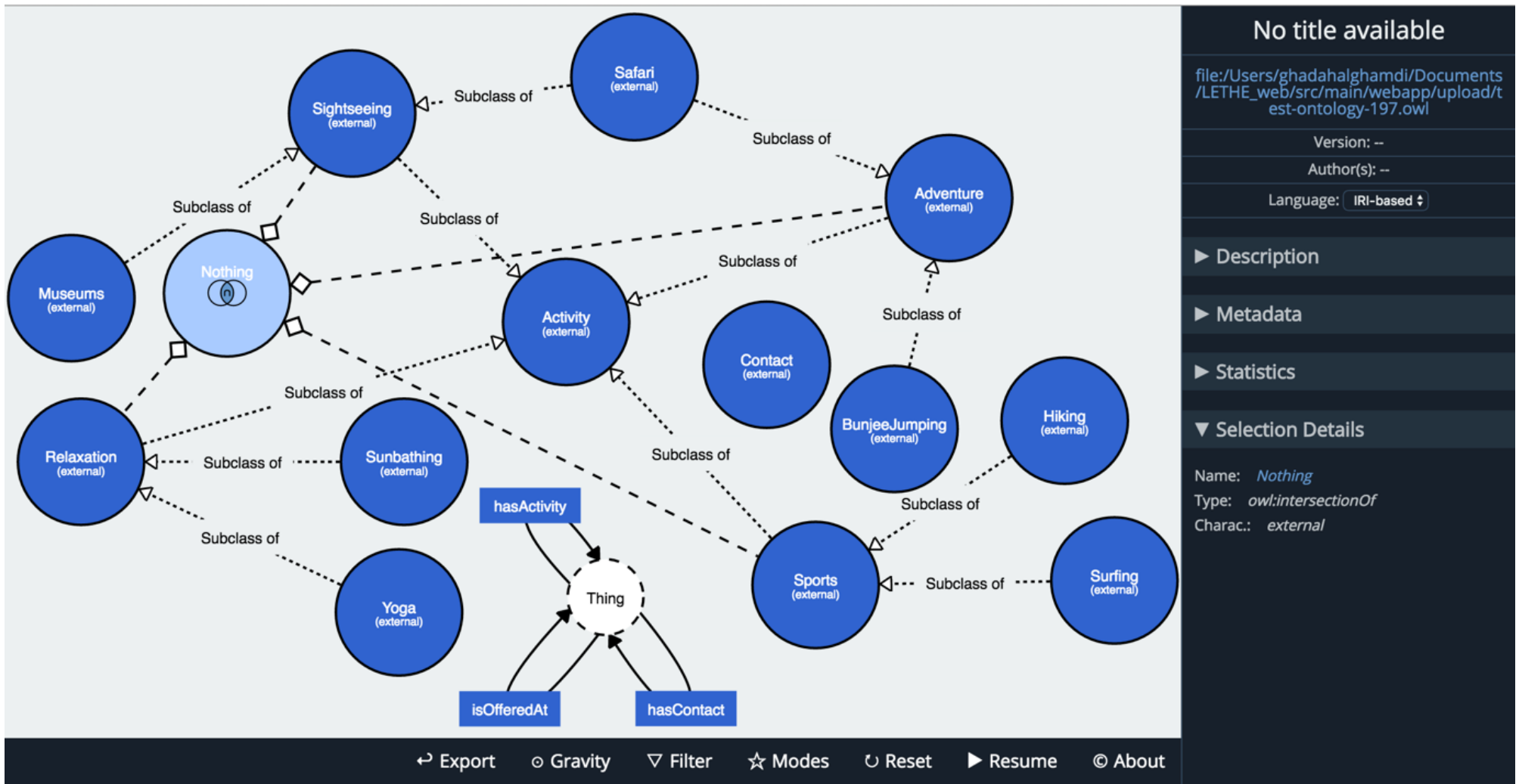


Figure 6-10: webVOWL visualisation of the resulting ontology based on ALCH TBoxes forgetting method (forgetting mode)

Second, the SHQ TBoxes forgetting method was chosen. The resulting ontology included five of the object properties that the travel ontology has, which were **hasActivity**, **hasAccommodation**, **hasPart**, **hasContact** and **isOfferedAt**. This is because of the aforesaid reason of this forgetting method not supporting role forgetting. However, it has forgotten the property **hasRating**, as this was a side effect of forgetting the concepts *Accommodation* and *AccommodationRating*, which are symbols that appear in all axioms that contain **hasRating**.

$$\begin{aligned} & \exists \text{hasRating}.\top \sqsubseteq \text{Accommodation} \\ & \top \sqsubseteq \forall \text{hasRating}.\text{AccommodationRating} \end{aligned}$$

The view also included the concepts that relate to *Activity* class such as *Relaxation* and its sub-class *Yoga*, and *Sightseeing* and its sub-classes *Safari* and *Museums*, as well as the *Contact* class. The resulting ontology is shown in Figure 6-11 in a readable format. The graph of the resulting ontology is illustrated in Figure 6-12.

Resulting Ontology

```

(Adventure ⊓ Relaxation) ⊑ ⊥
(Adventure ⊓ Sightseeing) ⊑ ⊥
(Adventure ⊓ Sports) ⊑ ⊥
(Relaxation ⊓ Sightseeing) ⊑ ⊥
(Relaxation ⊓ Sports) ⊑ ⊥
(Sightseeing ⊓ Sports) ⊑ ⊥
Adventure ⊑ Activity
BunjeeJumping ⊑ Adventure
Hiking ⊑ Sports
Museums ⊑ Sightseeing
Relaxation ⊑ Activity
Safari ⊑ Adventure
Safari ⊑ Sightseeing
Sightseeing ⊑ Activity
Sports ⊑ Activity
Sunbathing ⊑ Relaxation
Surfing ⊑ Sports
Yoga ⊑ Relaxation
trans(hasPart)
∃hasAccommodation.⊤ ⊑ (∃hasAccommodation.⊤ ⊔ ≤1hasActivity.⊤)
∃hasAccommodation.⊤ ⊑ (≥2hasActivity.⊤ ⊔ ≤1hasActivity.⊤)
∃hasContact.⊤ ⊑ Activity
∃isOfferedAt.⊤ ⊑ Activity
⊤ ⊑ ∀hasActivity.Activity
⊤ ⊑ ∀hasContact.Contact

```

Figure 6-11: The resulting ontology based on SHQ TBoxes forgetting method (forgetting mode)

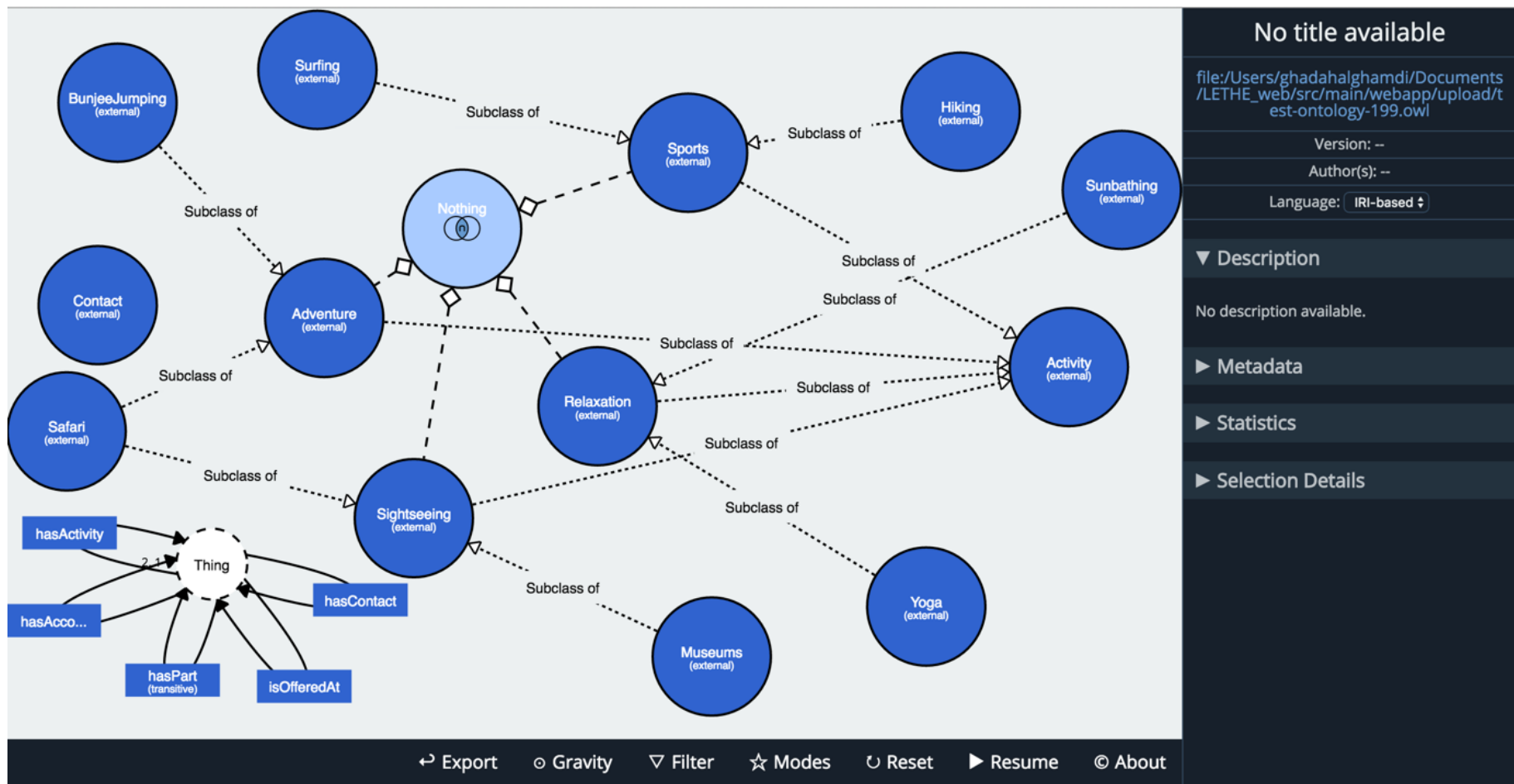


Figure 6-12: webVOWL visualisation of the resulting ontology based on SHQ TBoxes forgetting method (forgetting mode)

Third, the ALC with ABoxes forgetting method was chosen. Most of the axioms in the resulting ontology are similar to the result of ALCH TBoxes forgetting method. In addition, it included ABoxes axioms that involve individuals. Some of them are **\exists hasActivity.Hiking(BlueMountains)** and **\exists hasActivity.Museums(Sydney)**. The axioms mean that there is a hiking activity in *BlueMountains* and there is a museum activity in *Sydney*. The ontology also included all of the individuals as part of the *Thing* class, which can be seen in axioms such as **\top (BlueMountains)**, **\top (FourSeasons)** and **\top (Woomera)**. This can be useful for determining the different individuals in the ontology. Figure 6-13 illustrates the resulting ontology in a readable format. The visualisation of the obtained ontology is shown in Figure 6-14.

Resulting Ontology

```

(Adventure  $\sqcap$  Relaxation)  $\sqsubseteq$   $\perp$ 
(Adventure  $\sqcap$  Sightseeing)  $\sqsubseteq$   $\perp$ 
(Adventure  $\sqcap$  Sports)  $\sqsubseteq$   $\perp$ 
(Relaxation  $\sqcap$  Sightseeing)  $\sqsubseteq$   $\perp$ 
(Relaxation  $\sqcap$  Sports)  $\sqsubseteq$   $\perp$ 
(Sightseeing  $\sqcap$  Sports)  $\sqsubseteq$   $\perp$ 
Adventure  $\sqsubseteq$  Activity
BunjeeJumping  $\sqsubseteq$  Adventure
Hiking  $\sqsubseteq$  Sports
Museums  $\sqsubseteq$  Sightseeing
Relaxation  $\sqsubseteq$  Activity
Safari  $\sqsubseteq$  Adventure
Safari  $\sqsubseteq$  Sightseeing
Sightseeing  $\sqsubseteq$  Activity
Sports  $\sqsubseteq$  Activity
Sunbathing  $\sqsubseteq$  Relaxation
Surfing  $\sqsubseteq$  Sports
Yoga  $\sqsubseteq$  Relaxation
 $\exists$ hasActivity.Hiking(BlueMountains)
 $\exists$ hasActivity.Hiking(Warrumbungles)
 $\exists$ hasActivity.Museums(Canberra)
 $\exists$ hasActivity.Museums(Sydney)
 $\exists$ hasContact. $\top$   $\sqsubseteq$  Activity
 $\exists$ isOfferedAt. $\top$   $\sqsubseteq$  Activity
 $\top$   $\sqsubseteq$   $\forall$ hasActivity.Activity
 $\top$   $\sqsubseteq$   $\forall$ hasContact.Contact
 $\top$ (BlueMountains)
 $\top$ (BondiBeach)
 $\top$ (Cairns)
 $\top$ (Canberra)
 $\top$ (CapeYork)
 $\top$ (Coonabarabran)
 $\top$ (CurrawongBeach)
 $\top$ (FourSeasons)
 $\top$ (OneStarRating)
 $\top$ (Sydney)
 $\top$ (ThreeStarRating)
 $\top$ (TwoStarRating)
 $\top$ (Warrumbungles)
 $\top$ (Woomera)

```

Figure 6-13: The resulting ontology based on ALC with ABoxes forgetting method (forgetting mode)

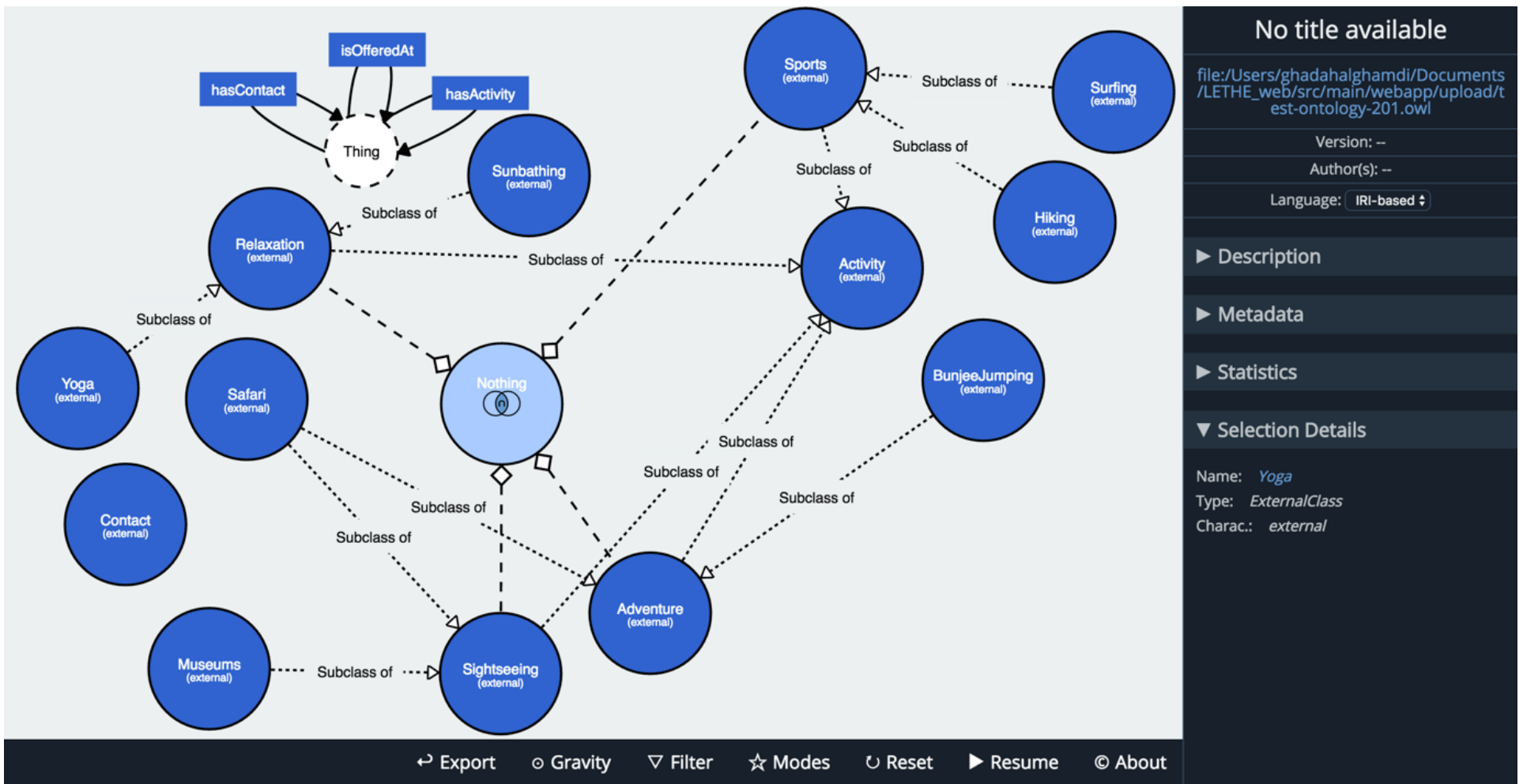


Figure 6-14: webVOWL visualisation of the resulting ontology based on ALC with ABoxes forgetting method (forgetting mode)

Table 4 summarises the resulting axioms of applying the forgetting mode on the travel ontology. These axioms are the results of using all of the forgetting methods. The axioms were divided to two categories, similar axioms among the results of all of the forgetting methods and noticeably different ones.

Forgetting Method	Similar resulting axioms among all forgetting methods	Resulting noticeably different axioms
ALCH TBoxes	- (Adventure \sqcap Relaxation) $\sqsubseteq \perp$	None
SHQ TBoxes	- (Adventure \sqcap Sightseeing) $\sqsubseteq \perp$ - (Adventure \sqcap Sports) $\sqsubseteq \perp$ - (Relaxation \sqcap Sightseeing) $\sqsubseteq \perp$ - (Relaxation \sqcap Sports) $\sqsubseteq \perp$ - (Sightseeing \sqcap Sports) $\sqsubseteq \perp$	- trans(hasPart) - \exists hasAccommodation.T \sqsubseteq (\exists hasAccommodation.T \sqcup ≤ 1 hasActivity.T) - \exists hasAccommodation.T \sqsubseteq (≥ 2 hasActivity.T \sqcup ≤ 1 hasActivity.T)
ALC with ABoxes	- Adventure \sqsubseteq Activity - BunjeeJumping \sqsubseteq Adventure - Hiking \sqsubseteq Sports - Museums \sqsubseteq Sightseeing - Relaxation \sqsubseteq Activity - Safari \sqsubseteq Adventure - Safari \sqsubseteq Sightseeing - Sightseeing \sqsubseteq Activity - Sports \sqsubseteq Activity - Sunbathing \sqsubseteq Relaxation - Surfing \sqsubseteq Sports - Yoga \sqsubseteq Relaxation - \exists hasContact.T \sqsubseteq Activity - \exists isOfferedAt.T \sqsubseteq Activity - T \sqsubseteq \forall hasActivity.Activity - T \sqsubseteq \forall hasContact.Contact	- \exists hasActivity.Hiking(BlueMountains) - \exists hasActivity.Hiking(Warrumbungles) - \exists hasActivity.Museums(Canberra) - \exists hasActivity.Museums(Sydney) - T(BlueMountains) - T(BondiBeach) - T(Cairns) - T(Canberra) - T(CapeYork) - T(Coonabarabran) - T(CurrawongBeach) - T(FourSeasons) - T(OneStarRating) - T(Sydney) - T(ThreeStarRating) - T(TwoStarRating) - T(Warrumbungles) - T(Woomera)

Table 4: Summary of the resulting axioms of all forgetting methods (forgetting mode)

The table shows that the ALCH TBoxes forgetting method did not compute any axioms that are not included in the results of SHQ TBoxes and ALC with ABoxes forgetting methods. The axioms are TBox axioms that define classes and properties related to *Activity* symbols. The result of applying SHQ TBoxes included axioms that contained the transitive property **hasPart** and cardinality restrictions. Finally, ALC with ABoxes forgetting method gave results that include ALC TBoxes along with ABoxes which consists of classes and properties that have members.

We can conclude from the results obtained by using uniform interpolation and forgetting functionalities that they cope with **LETHE**'s expected outcomes. The forgetting method for

ALCH TBoxes gave us results that involve TBoxes for the expressive language ALCH. In addition, the forgetting method for SHQ TBoxes gave results that are more expressive and detailed than ALCH and ALC languages. Finally, the application of ALC with ABoxes forgetting method has resulted in axioms that involve members, since this method interpolate for ALC TBoxes as well as knowledge bases (ABoxes). In addition, webVOWL representation of the restricted view ontology makes it easy for the user to distinguish the different concepts that corresponds with a certain forgetting method. This is shown by navigating the ontology graph, and the statistics section give information about object properties and individuals numbers.

6.2 Logical Differences Case Study

This case study was conducted to evaluate one of the uniform interpolation applications, which is logical differences. Logical differences functionality seeks to compute the differences between two ontologies. The user can choose to compute the logical differences between two ontologies either over common symbols that are shared between them, or specified ones. The computation of logical differences is done by reasoners which determine the entailments of axioms. The axioms that are not entailed by the first ontology are determined as new entailments among the two ontologies. The purpose of the function is to identify any entailments of an ontology that does not follow from the other one. In this case, we are interested in the entailments of the second (old version) ontology that do not follow from the first (new version) one.

In order to show the usefulness of using logical differences functionality, the Semantic Web for Research Communities (SWRC) ontology [50] was used. The ontology was developed in order to structure the research communities' concepts and their related terms including publications and bibliographical information. The SWRC ontology was first developed using various description logic languages including DAML+OIL and RDF(S) languages. Then, they have redeveloped it in its recent versions using OWL format. The ontology consists of six main concepts, which are *Project*, *Topic*, *Event*, *Organisation*, *Person* and *Publication*. Figure 6-15 illustrates the structure of the main ontologies concepts [50]. The total number of the ontology concepts is 53, which are presented in a taxonomy, and 20 object properties, in

which 10 of them are used in inverse object properties. In addition, there are a number of annotation axioms that describe the ontology concepts.

The *Person* class describes human and it is conceptualised by sub-classes including *Student* and *Employee*. The *Event* class describes events such as *Conference* or *Lecture*, which are subclasses of *Event*. The *Topic* class describes some types of topics such as *ResearchTopic*, which is a subclass of the *Topic* class. The *Organisation* class models the abstract concept of an organisation that has departments through the use of the subclass *Department*, or the conceptualisation of different types of organisations such as *Enterprise*. The *Project* class describes some types of projects such as *DevelopmentProject* or *ResearchProject* which are subclasses of the *Project* class. The *Publication* class describes the different types of publications which are in correspondence with the BibTeX references type. Some of the subclasses that are under the *Publication* class are *Article*, *Book* and *InProceedings* [50].

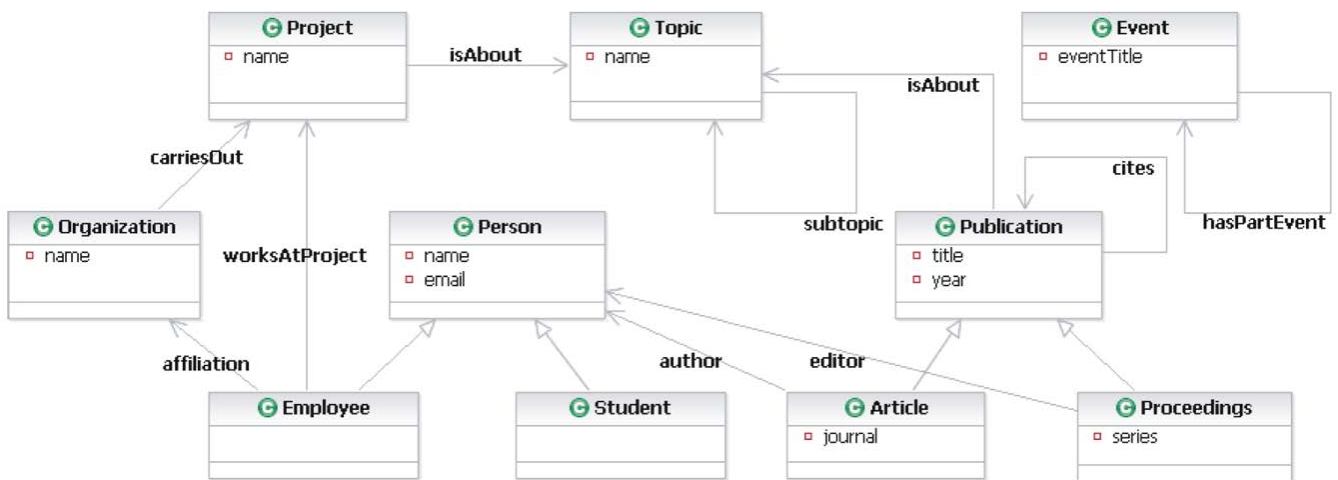


Figure 6-15: SWRC Ontology main concepts structure [50]

The ontology expressivity is OWL-DLP, in which DLP stands for Description Logics Programming [51]. OWL-DLP languages were designed to fill the gap between logic programming and an expressive semantic web language like description logics. According to Vrandečić [51], “it lies within the intersection of description logics and logic programming”. In addition, it provides the expressivity of OWL DL language and serves the purposes of

developing future semantic programmes [52]. It has been used in the development of the SWRC ontology for the benefits that it provides and for its sufficient expressivity of describing the ontology concepts.

The ontology has two different versions available in the ontology website [53]. For the purpose of evaluating the functionality of logical differences in our tool, the two versions were used. These versions are 0.3 and 0.7.1 and both of them are in the OWL format. The earlier version is 0.3, which lacks annotation axioms that help in providing description of the ontology axioms. Moreover, the newer version which is 0.7.1 included many improvements. If we compare it with the older version, we notice that new classes have been introduced in the recent version, such as *Document* class. The *Document* class is the super class of *Publication* and *Unpublished* classes. This is not the case with the earlier version (0.3) in which the *Unpublished* class is a subclass of *Publication*. The two versions included many other differences that can be determined using the logical differences functionality.

First, the old version ontology elements are visualised using webVOWL (Figure 6-16). This process is conducted in order to analyse the ontology and check its classes and properties through visualisation, which makes it easier for us to understand the differences between the two versions. The statistics section shows that the ontology contains 55 classes, 44 object properties, 30 datatype properties and no individuals.

Second, the new version ontology visualisation illustrates that there are 71 classes, 48 object properties and 46 datatype properties (Figure 6-17). The figure shows that the new ontology has major enhancements and changes regarding the introduction of new classes, and the creation of new object properties in which they both provide better conceptualisation to the SWRC ontology.

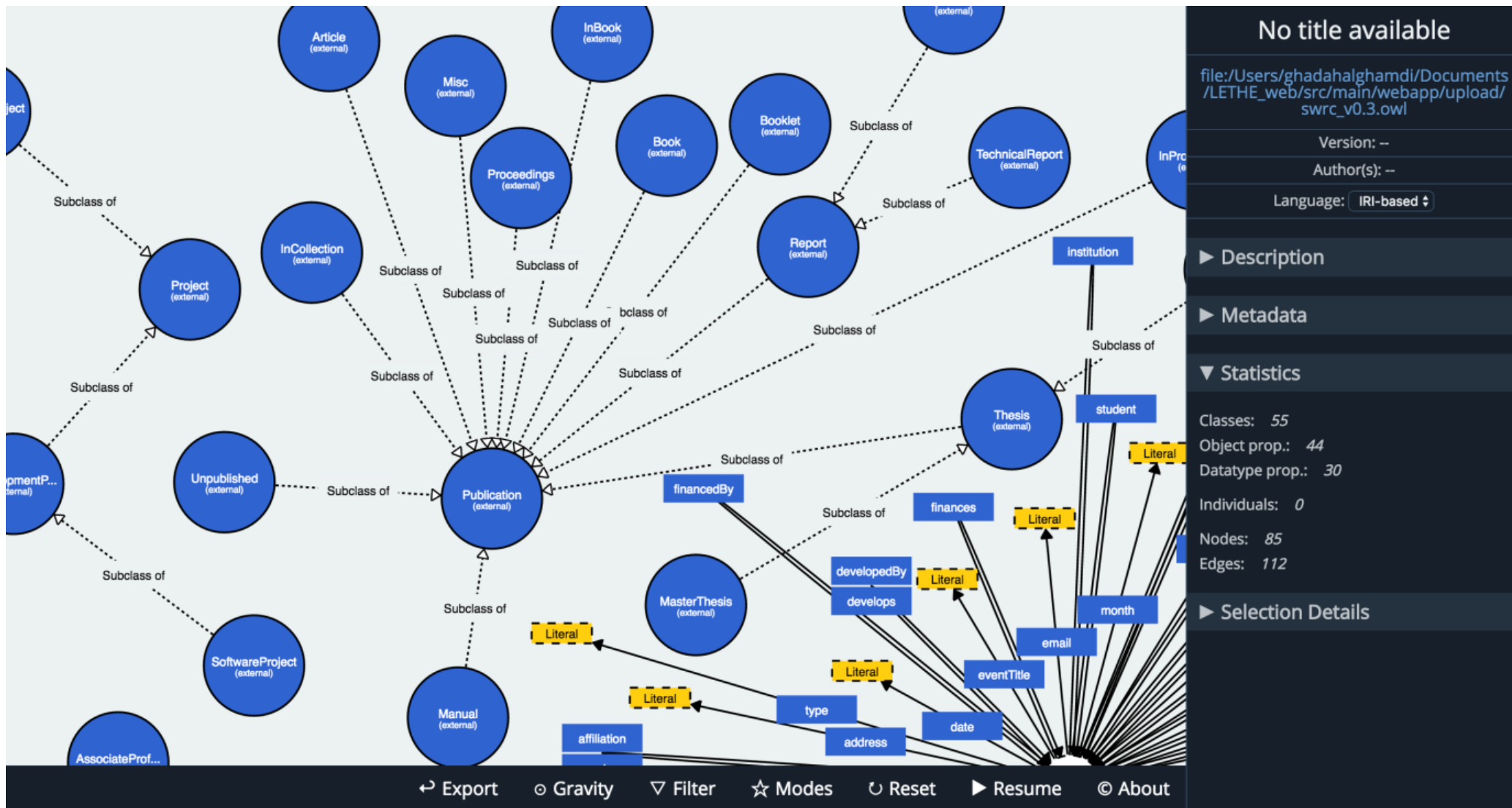


Figure 6-16: webVOWL visualisation of the SWRC ontology version 0.3

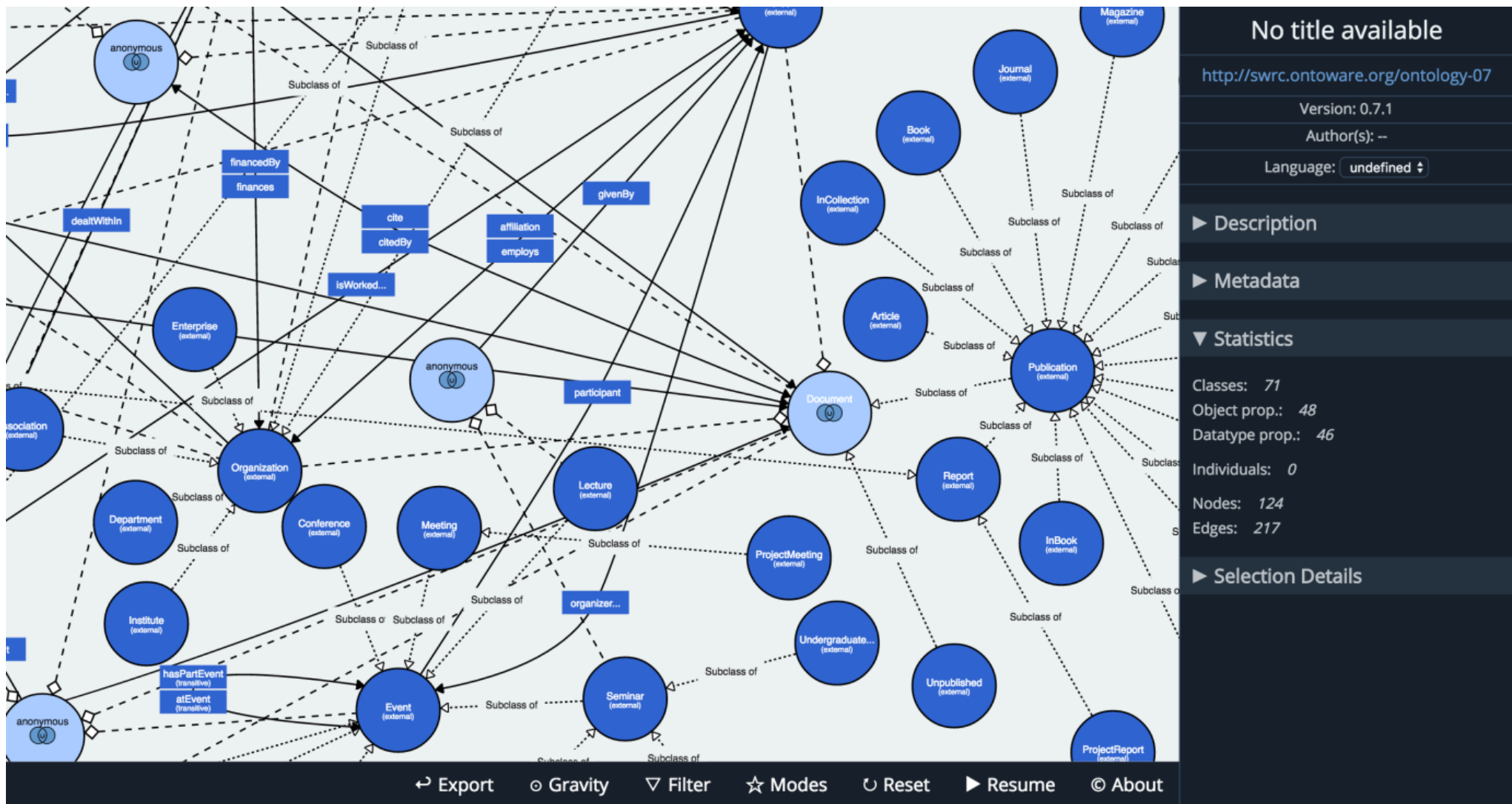


Figure 6-17: webVOWL visualisation of the SWRC ontology version 0.7.1

The result of applying the two ontologies in the system produced the difference between the two ontologies. The aim was to obtain the axioms that the old version contained in comparison with the new version. This is beneficial in tracking what changes the old version included and to determine the concepts that are not entailed by the new version. The input parameters that were inserted were as follows:

- First Ontology: SWRC Ontology version 0.7.1
- Second Ontology: SWRC Ontology version 0.3
- Logical Differences Option: Common Signatures
- Forgetting Method: SHQ TBoxes
- Approximation Level: 2

The parameter of the first ontology was chosen to be the new version, while the second ontology was chosen to be the older version. The reason for that is to obtain the axioms that are entailed by the older version and not by the new one. This was conducted to accomplish the aforementioned aim. Common Signatures was chosen as the logical differences option, in which specified signatures are not selected. This is useful to make **LETHE** computes the logical differences based on all common signatures that the two ontologies share. Changing the parameter of the forgetting method did not make differences in the result, as all of them gave the same resulting axioms. Figure 6-18 illustrates the result in readable format.

Resulted Axioms

```

AcademicStaff ⊆ ∀cooperateWith.AcademicStaff
Book ⊆ ∀editor.Person
InBook ⊆ ∀editor.Person
InCollection ⊆ ∀editor.Person
InProceedings ⊆ ∀editor.Person
Institute ⊆ ∀cooperateWith.Institute
Proceedings ⊆ ∀editor.Person
Project ⊆ ∀isAbout.ResearchTopic
SoftwareProject ⊆ ∀product.Product
Unpublished ⊆ Publication

```

Figure 6-18: The resulting ontology after applying logical differences function

The figure shows the axiom **Unpublished ⊆ Publication** in which it states clearly that *Unpublished* class is subclass of *Publication* class. This axiom exists in the old version ontology, which shows that **LETHE** gave the results of the concepts that are entailed by the

old version and not by the new one. Figure 6-19 shows the axiom in the old ontology version visualised in webVOWL.

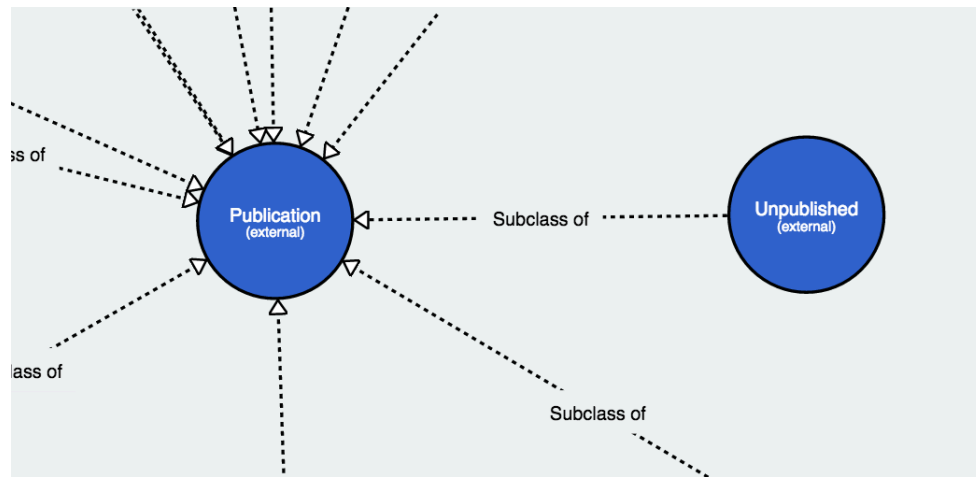


Figure 6-19: Part of the visualisation of the SWRC old version ontology showing the Unpublished class is a subclass of Publication

The following table was created manually to illustrate the different concepts between the two ontologies with the clarification of key changes that have occurred (Table 5).

Ontology element	Ontology element type	Version 0.3	Version 0.7.1
cooperateWith	Object property	AcademicStaff \sqsubseteq \forall cooperateWith.AcademicStaff	AcademicStaff \sqsubseteq \forall cooperateWith.Person
		Institute \sqsubseteq \forall cooperateWith.Institute	Organization \sqsubseteq \forall cooperateWith.Organization
editor	Object Property	Book \sqsubseteq \forall editor.Person InBook \sqsubseteq \forall editor.Person InCollection \sqsubseteq \forall editor.Person InProceedings \sqsubseteq \forall editor.Person Proceedings \sqsubseteq \forall editor.Person	\exists editor.T \sqsubseteq Person
isAbout	Object Property	Project \sqsubseteq \forall isAbout.ResearchTopic	Project \sqsubseteq \forall isAbout.Topic
product	Object Property	SoftwareProject \sqsubseteq \forall product.Product	No product property
Unpublished	product	Unpublished \sqsubseteq Publication	Unpublished \sqsubseteq Document

Table 5: Summary of the resulting axioms of logical differences function applied on the SWRC ontology

The table shows the axiom **AcademicStaff** \sqsubseteq \forall **cooperateWith.AcademicStaff** in the first row of the table that exist in the old version. This axiom contains *AcademicStaff* on the right hand side of the inclusion, which has been changed to *Person* class in the new version (**AcademicStaff** \sqsubseteq \forall **cooperateWith.Person**). In addition, the **editor** property was eliminated from most of the axioms in the new version, comparing to the old version, in which it exists in axioms including **Book** \sqsubseteq \forall **editor.Person**, **InBook** \sqsubseteq \forall **editor.Person** and **Proceedings** \sqsubseteq \forall **editor.Person**. Moreover, the right hand side of the axiom **Project** \sqsubseteq \forall **isAbout.ResearchTopic**, *ResearchTopic* class was changed to *Topic* class in the new version. The object property **product** was eliminated from the new version. Finally, the class *Unpublished* became sub-class of *Document* class, which is the parent class of *Publication* and *Unpublished* classes in the new version.

In conclusion, the previous table illustrates that logical differences functionality is useful and sufficient in extracting the differences between the two ontologies' versions. The first implementation of logical differences successfully produced results. Moreover, webVOWL was useful in navigating the ontology and clearly identified the differences between the different ontology versions. webVOWL has illustrated the graphical representation of ontologies properly with the inclusion of all the corresponding classes and object properties.

6.3 VOWL and OntoGraf Comparison

In order to illustrate webVOWL's capabilities in the perspective of other thorough developed visualisation tools, the following comparison was conducted. The study involved comparison between webVOWLS' visualisation features and OntoGraf tool features. The comparison was performed in terms of their layouts, node representation, types of files exportation and their platforms. The OntoGraf tool background can be returned to in Section 2.4, while webVOWL background is mentioned in Section 3.3.

OntoGraf provides different type of layouts including grid, radial, spring, tree, vertical directed or horizontal directed. Other than these layouts, the user can interact with the graph to place the nodes in different locations.

Figure 6-20 shows the travel ontology represented in OntoGraf in horizontal directed layout. The figure also shows the tooltip of the *NationalPark* class, which provides detailed information about the class. In contrast, webVOWL's layout provides the advantages of force-directed methods, using D3 libraries, which gives the user the ability to manipulate the graph in a dynamic way. As the graph in the layout moves dynamically while interacting with it, the motion can be paused with the pause controller to locate nodes steadily.

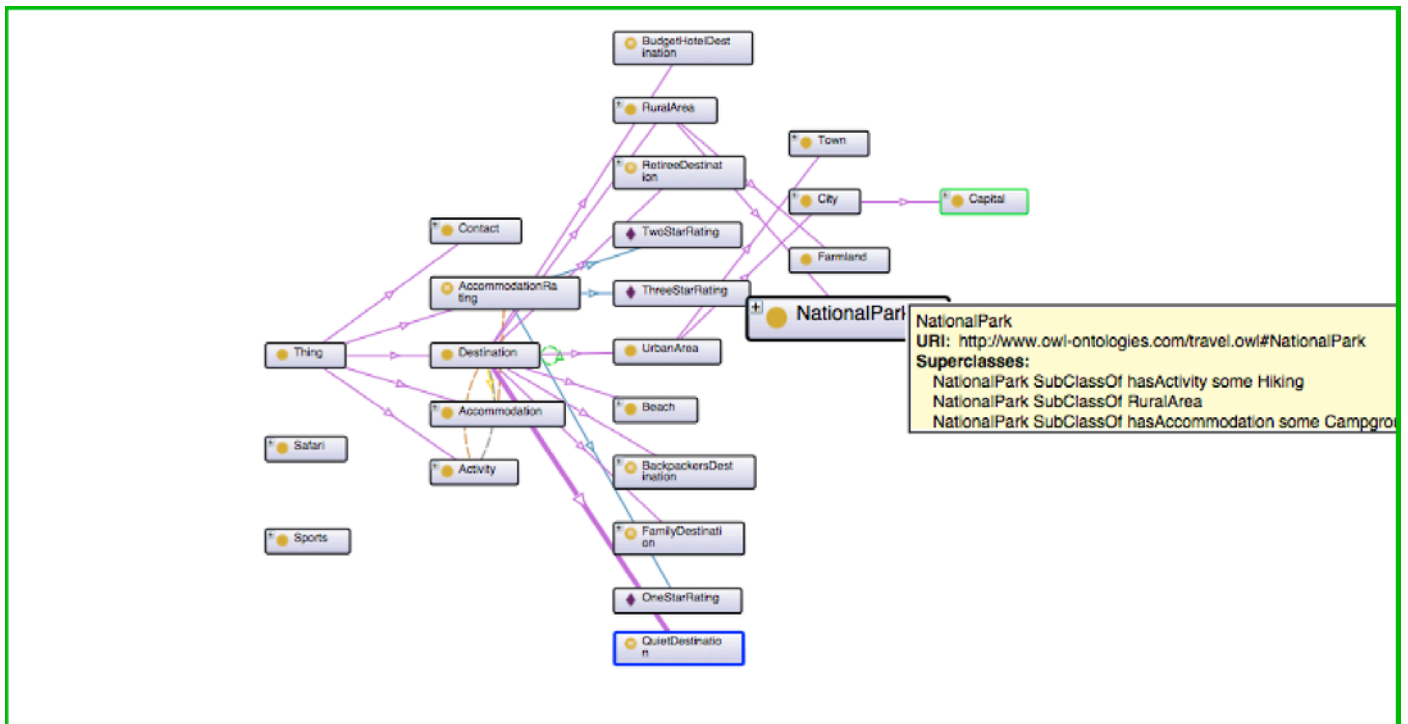


Figure 6-20: OntoGraf visualisation of the travel ontology

OntoGraf presents the relationship between ontology nodes (classes) as arrows with different colours that represents different types of relationships. These relationships are subclass, individual, domain and range of object properties, and equivalence. OntoGraf does not present object properties and datatype properties in separate nodes nor their characteristics, including deprecated, symmetric, functional or transitive properties. However, it present individuals as nodes to illustrate their belonging to a certain class. In comparison, webVOWL presents classes as circle nodes, properties as rectangles, and individuals as numbers in the centre of classes' nodes.

Moreover, OntoGraf provides tooltips of nodes that shows more details including URI, subclasses, disjoint classes and annotations. This tooltip is configurable in which the user can

add or remove detailed information about a certain node. On the other hand, webVOWL provides a selection details area which encloses information about the selected node in the graph. This information depends on the type of the selected node. For example, the information of the class node includes name, type, disjoints (with other classes), individuals and comments. A property node has different information which is name, type (object or datatype property), domain and range.

OntoGraf provides the capability to export the ontology graph to images in various extensions, which are PNG, JPG and GIF. In addition, the graphs can be exported to a DOT (Graph Description Language) file, which can be customised using different JavaScript libraries [55, 56]. On the second hand, webVOWL provides exporting of graphs to SVG. Scalable Vector Graphics (SVG) is an XML-based language that features the capability of scaling images without losing the quality of them [57]. The XML language in SVG helps in editing graphs or customising the specifications of them easily. In addition, webVOWL provides the option of exporting the resulting graph to JSON file.

OntoGraf was developed as a plugin to Protégé editor, and was implemented using Java programming language. This suits standalone applications that are based on Java language. On the other hand, webVOWL was developed completely using open web standards, including HTML, CSS and JavaScript. Thus, webVOWL is suitable for applications that use the aforementioned web languages.

This comparison study shows that webVOWL provides dynamic interaction layout, in which the user can manipulate the graph and customise its view according to their needs. Moreover, VOWL specifications of visualising the OWL language are rich with many different types of OWL elements. Table 6 shows that VOWL presents classes, equivalent classes, subclasses properties and datatypes. In contrast, OntoGraf presents only classes and individuals as nodes. Thus, visualisation by VOWL language in webVOWL is sufficient for non-expert users to visualise ontologies and identify the relationship between their elements. In addition, it provides enough details for the expert users. According to Lohmann [31], positive evaluation outcomes were obtained from expert users who have used webVOWL for the purpose of evaluating the tool. One noteworthy point of the evaluation outcomes is the expert users' praise for force-directed layout, which helped them to easily identify the hierarchical relationships between the ontology nodes.

Comparison points	webVOWL	OntoGraf
Export to	SVG, JSON	Image (PNG,JPG,GIF), DOT
Platform	Web	Local Applications
Layout	Dynamic force-directed graph layout using the JavaScript library D3.	Different layouts such as: Grid (alphabetical), Radial, Spring, tree, vertical directed, or horizontal directed.
Node representation	Classes, equivalent classes, subclasses, properties, datatypes.	Classes, individuals
Implementation language	HTML, CSS, SVG and JavaScript.	Java

Table 6: Summary of the comparison between webVOWL and OntoGraf

6.4 Our Tool and PATO Tool Comparison

This section presents a comparison between one of the previously mentioned modularisation tools, **PATO**, and our system’s uniform interpolation functionality in terms of their input parameters, segmentation methods and the computed ontologies.

The **PATO** tool is a standalone application developed as an implementation of the partitioning modularisation approaches. These approaches aim to produce modules of ontologies based on their structural content. In addition, the steps to produce the modules involve the use of another tool, Pajek [58, 59]. Pajek is a network analysis tool used to create dependency graphs that are used as input parameters in **PATO**. On the other hand, our system uses saturation-based reasoning methods that are useful in preserving the logical entailments of ontologies. In other words, these methods produce restricted view ontologies based on ontologies’ semantics. Section 2.4.1 provides the background of the **PATO** tool.

The process of producing modules using **PATO** involves the following input parameters: an ontology file in OWL format, a network file that represents the dependency graph of .net extension, several clustering options such as “include subclass links”, “include property links”, or “include definition links”, island or cluster size and an output directory. Clicking on the convert button in the **PATO** interface converts the input ontology file to a .net file, which

contains the relations between the ontology entities. Figure 6-21 illustrates this step using the travel ontology. The methods used to produce modules are based on the frail connections among these modules. Another important parameter is the size of the island or module. This parameter is inserted after the ontology is converted to a .net file. The user should also insert the output directory in which the resulting modules are saved. In contrast, the function of uniform interpolation in our tool requires three basic parameters needed to produce restricted view ontologies. The first one is the ontology file in OWL or RDF format. The second one is the forgetting method, which can be one of three methods (ALCH TBoxes, SHQ TBoxes and ALC with ABoxes). The third parameter is a set of symbols, which defines the outcome's concepts.

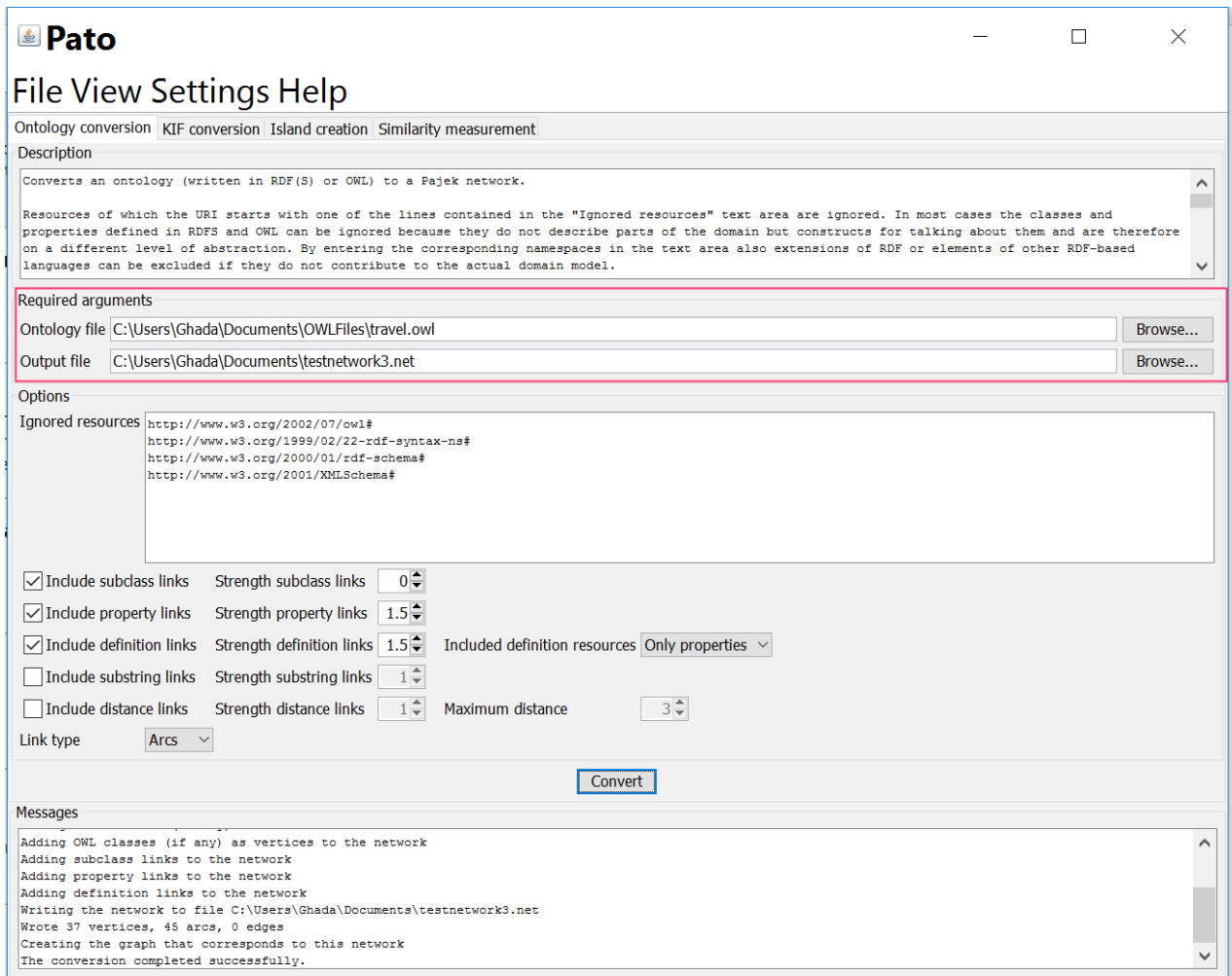


Figure 6-21: The interface of *PATO* illustrating the step of inserting an ontology and the .net file before the conversion process

PATO produced .net extension modules that can be realised with network analyser tools such as Pajek. According to [18], these modules are “self-contained OWL ontologies”. However, the testing of **PATO** did not result in OWL files; rather they are .net files and a .clu that represent a partition file. In comparison, our tool produced a .owl file in OWL/XML syntax, which can be used with most ontology editors including Protégé, SWOOP or TopBraid.

Figure 6-22 displays all of the modules contained in the (partition) .clu file produced using Pajek, with elements of each module represented by different colours. For example, the elements of one of the modules that represents destination hotels are coloured red. The elements of this module include **LuxuryHotel**, **Hotel**, **BedAndBreakfast**, **Campground** and **AccommodationRating**. This example shows how the tool has formed a module of hotels and related terms from the links forming these terms (concepts) in the original ontology.

Moreover, by looking at the module represented in green, the majority of the elements are related to the *Destination* class. These elements are subclasses of the *Destination* class, which are *Beach*, *BudgetHotelDestination*, *City*, *UrbanArea*, *RuralArea*, *NationalPark*, *Town*, and *Farmland*. The result also included the *hasAccommodation* property. Thus, we can deduce that the computed module is related to the destination concepts, as the majority of the classes are subclasses of the *Destination* class. However, the original ontology involves other subclasses of the *Destination* class that were not included as part of the resulting module. This could be because of the methodology used to produce the modules, which is structuring-based partitioning that produce modules based on the weak connections among them. To compare the results given by **LETHE** to those obtained by **PATO**, the set of symbols that were selected in our tool are similar to the **PATO** results. Figure 6-23 shows the resulting ontology visualised in webVOWL. The figure shows that **LETHE** included two classes that determine the logical entailment of symbols in the ontology, which are *Thing* and *Nothing* classes. The *Nothing* class were included to establish that the *RuralArea* and *UrbanArea* classes negate each other or are disjoint ($\mathbf{RuralArea} \cap \mathbf{UrbanArea} \sqsubseteq \perp$). This axiom clearly shows that **LETHE** has preserved the semantics of the original ontology by including the required classes (*Thing* and *Nothing*). Moreover, with **LETHE**, the concepts of the resulting view can be easily determined because of the ability to select symbols that should be included in the view. Although it could require effort to select the symbols manually, particularly if the ontology is large, a future work on this matter is described in Chapter 7.

:\\Users\\Ghada\\Documents\\testnetwork3.net (37) / C2. C:\\Users\\Ghada\\Documents\\Output-PATO-travel-OWL3\\index.clu (37)

t Layers GraphOnly Previous Redraw Next Options Export Spin Move Info FishEye Wait

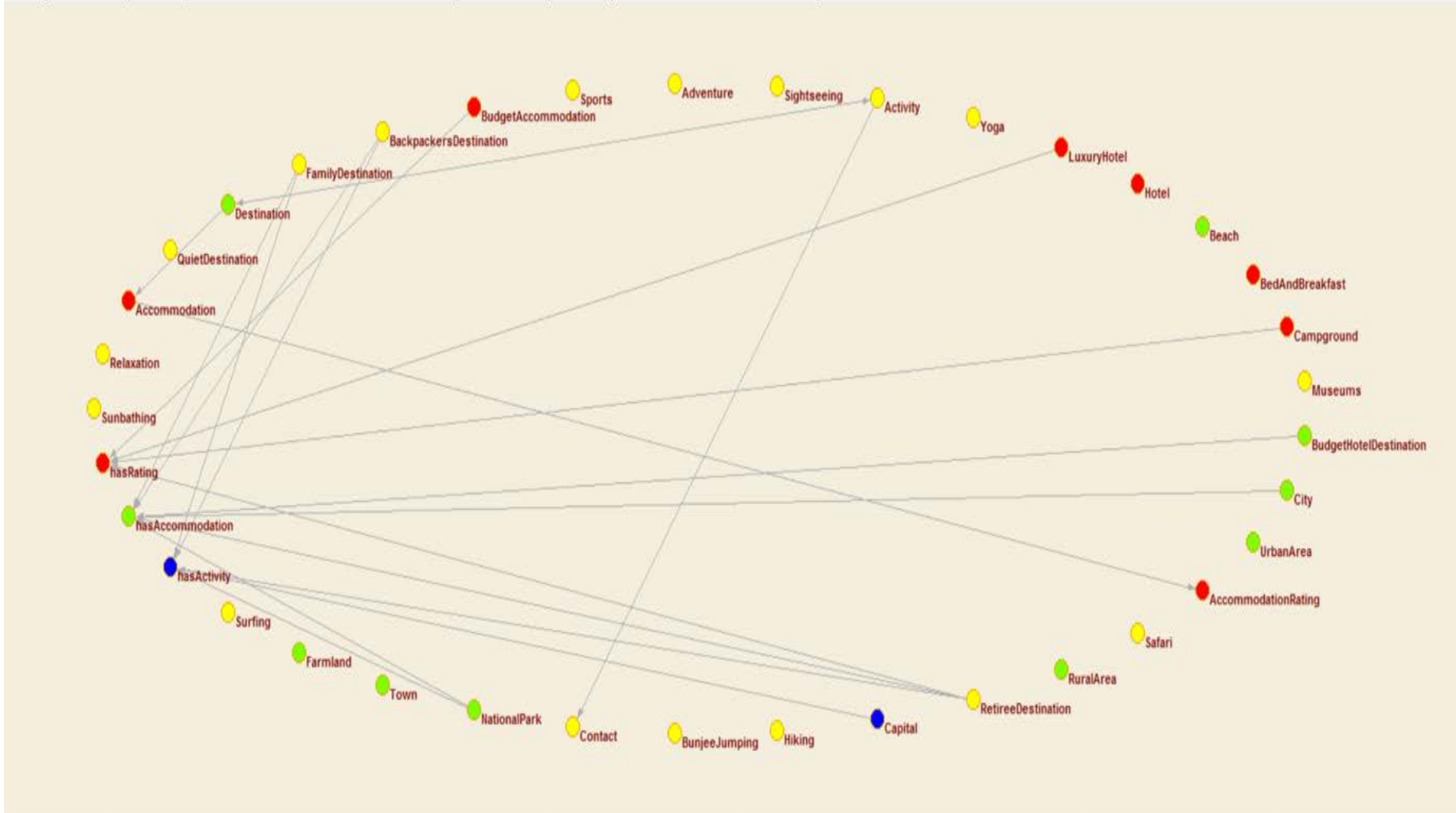


Figure 6-22: Pajek visualisation of the resulting .clu (partition) file, illustrating the produced modules in different colours

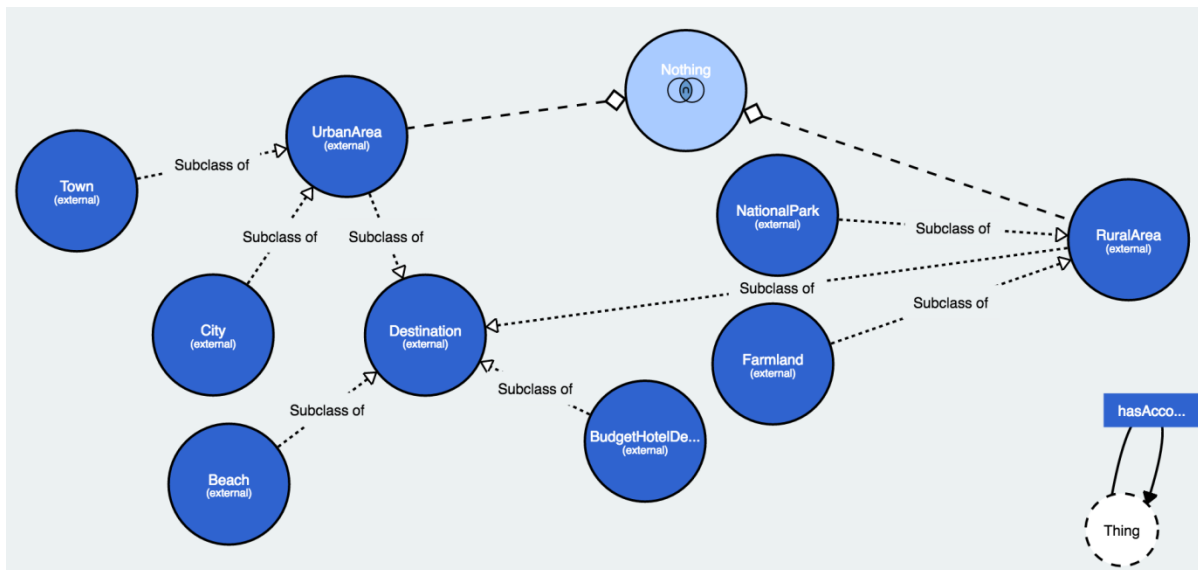


Figure 6-23: wevVOWL visualisation of the resulting restricted view ontology

The content of .net files that are produced by **PATO** can be browsed by opening it using a text editor. Figure 6-24 shows the content of one of the modules that was produced from the travel ontology. The figure also shows that it is difficult to obtain the logical meaning of the presented output, as it only describes the existence of 10 vertices, which are the previously mentioned *Destination* classes, and the arcs, which describes the relationship “is-a” between these classes. On the other hand, using our tool to browse the resulting ontologies makes it easier to understand the logical meaning of the axioms. Figure 6-25 shows the resulting ontology in a readable format.

```

/Volumes/BOOTCAMP/Users/Ghada/Documents/O
1  |*Vertices 10
2  1 "Destination" ellipse
3  2 "Beach" ellipse
4  3 "BudgetHotelDestination" ellipse
5  4 "City" ellipse
6  5 "UrbanArea" ellipse
7  6 "RuralArea" ellipse
8  7 "NationalPark" ellipse
9  8 "Town" ellipse
10 9 "Farmland" ellipse
11 10 "hasAccommodation"
12 *Arcs
13 2 1 NaN l "isa"
14 1 2 0.0000000 l "isa"
15 4 5 0.0000000 l "isa"
16 5 4 NaN l "isa"
17 6 1 NaN l "isa"
18 1 6 0.0000000 l "isa"
19 5 1 NaN l "isa"

```

Figure 6-24: Extract of .net file syntax

Resulting Ontology

```

(RuralArea ⊓ UrbanArea) ⊆ ⊥
Beach ⊆ Destination
BudgetHotelDestination ⊆ Destination
City ⊆ UrbanArea
Farmland ⊆ RuralArea
NationalPark ⊆ RuralArea
RuralArea ⊆ Destination
Town ⊆ UrbanArea
UrbanArea ⊆ Destination
∃hasAccommodation.⊤ ⊆ Destination

```

Figure 6-25: The resulting ontology in readable format

In conclusion, the **PATO** tool can be used to produce clusters (modules) in cases in which the user likes to segment a large ontology when they do not have an in-depth understanding of its content. Moreover, the segmentation of such large ontologies produces ontologies that are smaller, making it easier to browse their content. The user then can use our tool to obtain a restricted view ontology based on a set of related concepts (symbols). However, there is a gap between the use of the output produced by **PATO** and applying these outputs that are of type .net and .clu on our tool. As our tool only works with OWL or RDF formats, this is an issue that can be addressed with an enhanced version of **PATO** that can produce modules of OWL or RDF formats. Table 7 summarises the comparison points between **PATO** and our tool.

Comparison points	PATO	Our Tool
Parameters	<ul style="list-style-type: none"> - Ontology file - Network file - Clustering options including subclass links, property links, and definition links. - Island (cluster) size - Output directory 	<ul style="list-style-type: none"> - Ontology file - Forgetting method - Symbols
Segmentation methods	Modularisation: partitioning approaches	Saturation-based reasoning methods
Resulting output	Modules in type of networks (.net) or partitions (.clu)	OWL/XML syntax file

*Table 7: Summary of the comparison between **PATO** and our tool*

6.5 Summary

The evaluation of the tool was conducted in the form of case studies that aims to illustrate results given by **LETHE** functionalities. The chapter included two main case studies that were performed with regard to uniform interpolation, forgetting and logical differences functionalities. The results were visualised using webVOWL to evaluate the graphs of the resulting ontologies. A comparison between VOWL and OntoGraf was presented to illustrate the capabilities that VOWL provides. Lastly, Section 6.4 provided a comparison between **PATO** and our tool.

7 Conclusions and Future Work

Recently, the number of real world applications that use ontologies to describe their terms and concepts has increased. These ontologies are often quite large in order to fit such applications. This has led to the need for tools that can extract smaller ontologies from larger ones. Many approaches have been introduced for this purpose. One of these approaches is modularisation. Modularisation approaches segment ontologies into smaller modules. The modularisation approaches vary in their types. Some of these types are partitioning approaches and module extraction approaches, which were discussed in Chapter 2, Section 2.4. Most of the methods that are used in these approaches are structural rather than semantical. On the other hand, the methods used in **LETHE**, a tool developed by Koopmann, are semantical, which preserves the logical outcomes of ontologies.

The main goal of the current project was to exploit the capabilities provided by **LETHE** in a web-based browser with advanced analysis features. Through the browser, the user can gain an in-depth understanding of the logical structure of ontologies and the relationship between their terms.

In our project, we have proposed a web-based tool that exploits the functionalities provided by **LETHE**. These functionalities are implementations of uniform interpolation, logical differences and TBox abduction. Along with exploiting such features, visualisation features were integrated. The web tool targets expert and non-expert ontology users who would like to browse ontologies and extract restricted views out of them. The expert user is able to use the restricted view ontologies for debugging purposes, ontology analysis and reuse. The non-expert users can exploit the tool in obtaining smaller views of ontologies, which makes it easier for them to understand their contents.

In our tool, we tried to exploit all of the functionalities provided by the **LETHE** library. However, only a complete implementation of uniform interpolation and logical differences functionalities were provided. The implementation of TBox abduction was only partially implemented due to the effort and time it requires to completely implement it. Moreover, TBox abduction functionality in the current version of **LETHE** is only supported for ALCH

expressive language and only for acyclic TBoxes. Thus, its implementation would not support most ontologies that could be written in more expressive languages, such as SHQ. Table 8 summarises the features provided by our tool’s interface in comparison with the **LETHE** standalone version interface. These features include the three functionalities provided by **LETHE**, browsing ontologies in a readable format, symbols filtration, visualisation capabilities and downloading ontologies. In the **LETHE** standalone version, uniform interpolation is implemented along with ontology browsing in a readable format and the downloading of the computed ontologies. On the other hand, the **LETHE** web version provides the full implementation of uniform interpolation and logical differences, but only partial implementation of TBox abduction. In addition, symbols filtration is provided, which allows the users to filter symbols to make it easier for them to select symbols. Visualisation features are provided and the resulting ontologies can be downloaded.

Features	LETHE standalone version	LETHE web version
Uniform interpolation	Implemented without the forgetting mode	Implemented with the forgetting mode
Logical differences	Not implemented	Implemented
TBox abduction	Not implemented	Partially implemented
Ontology browsing in readable format	Implemented for uniform interpolation	Implemented for uniform interpolation and logical differences
Symbols filtration	Not implemented	Implemented
Visualisation	Not implemented	Visualise original and resulting ontologies
Downloading resulting ontologies	Implemented for uniform interpolation	Implemented for uniform interpolation and logical differences

*Table 8: Summary of the features provided by the **LETHE** standalone version and the **LETHE** web version*

Due to the deadline required to submit the project, some of its objectives were not met, which are as follows:

- Determine the size of ontology that **LETHE** can handle and identify bottlenecks.
- Apply the analyser to a real-life ontology in medicine or bio-informatics.

- Provide an artefact within the scope of the application (such as a manual) as a resource for learning how to use **LETHE**.

Nonetheless, these objectives are secondary ones, as the main purpose of the project was achieved, which is developing a web application that exploits the **LETHE** tool and involves analysis visualisation features. Moreover, one of the project objectives is to evaluate the project against different types of ontologies. This was achieved, as the tool has been tested against travel, bibtex and SWRC ontologies. The testing of the bibtex ontology was performed as part of illustrating the use cases of the uniform interpolation functionality, which was provided in Chapter 5, Section 5.2. The testing of the travel and SWRC ontologies can be referred to in the case studies that were presented in Chapter 6, Sections 6.1 and 6.2.

The tool is a first prototype and has limitations, some of which are fundamental. These limitations are concerned with weaknesses of uniform interpolation. One limitation is the **LETHE**'s inability to support the forgetting of roles in the SHQ TBoxes forgetting method. In addition, **LETHE** lacks the support of nominals that can be included in TBox axioms. These limitations can be tackled in future improvements of the **LETHE** library. Other limitations are concerned with testing a real-world ontology with our tool. The terms in the ontology including classes and object properties were identified using concept IDs, while their nominal presentation (objects names) was performed using `rdfs:label` element. The presentation of such classes and object properties to the user would have no meaning, as they are numeric identifiers. However, this can be solved by presenting their corresponding `rdfs:label` to the user in order for them to select a set of symbols based on them. As **LETHE** only accepts symbols of type `OWLEntity`, the tool in the current version cannot deal with `rdfs:label` to compute ontologies based on the `rdfs:label` type. This problem can be tackled by converting `rdfs:label` to `OWLEntity` type.

Another restriction of our tool is concerned with the logical differences service. The service successfully produces the new entailments among two different versions of ontologies. However, a table of the differences based on the new entailments has to be produced automatically, showing specific details, such as the type of the entity that was changed, or showing the symbols that have differed from one version to another. The table can be similar to the one provided in the logical differences case study in Chapter 6, Section 6.2.

The tool also cannot semi-automatically select symbols. For instance, if the user selects a certain symbol that can be an object property or a class, the tool automatically selects the subsequent objects of the ontology to compute the uniform interpolation based on them. This feature would make it easy for users to obtain a restricted view based on a certain class or property hierarchy.

It was intended to provide a manual or an artefact that can be used as a resource for users to return to. This manual can provide a comprehensive description of our tools services as well as a user guide. The resource can be updated with future improvements in the tool. Nonetheless, this objective can be met in the future.

The following points summarises our tool's limitations:

- Supporting the TBox abduction service in the interface.
- Semi-automatic selection of ontology symbols.
- Converting `rdfs:label` to OWLEntity symbols to support the use of real world ontologies that uses numeric identifiers to identify their objects.
- Production of a detailed logical differences table that illustrates detailed description of the differences.

The use of the **LETHE** library in this project has proven its usefulness in integrating it in applications that use web-based technologies. In addition, we have proven that it is possible to use the library in ontology analysis applications to analyse ontologies of different expressive description languages including, ALC, ALCH and SHQ. This was illustrated in Chapter 6, where two case studies were conducted for the purpose of analysing the computed ontologies. Finally, as mentioned previously, the tool is a first prototype and it is open for future enhancements and improvements.

References

- [1] Ontology editors [Online] Available: https://www.w3.org/wiki/Ontology_editors [Accessed 29 Feb 2016]
- [2] Koopmann, Patrick A., and Renate A. Schmidt, "LETHE: Saturation-Based Reasoning for Non-Standard Reasoning Tasks." In *CEUR Workshop Proceedings* 1387, pp. 23-30, 2015.
- [3] Koopmann, Patrick A., "Practical uniform interpolation for expressive description logics." *PhD Theses*. Manchester, UK: The University of Manchester, 2015.
- [4] Gruber, Thomas R., "A Translation Approach to Portable Ontology Specifications." In *Knowledge Acquisition*, 5(2), pp. 199-220, 1993.
- [5] Gasevic, Dragan., Dragan Djuric, and Vladan Devedzic, "Ontologies." In *Model Driven Engineering and Ontology Development*, 2nd ed., pp. 45-80, Springer Publishing Company, Incorporated, 2009.
- [6] Turhan, Anni-Yasmin Y., "Introductions to Description Logics - A Guided Tour." In *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8067, pp. 150-61, 2013.
- [7] Shih-Wei Chen, Yu-Ting Tseng and Tsai-Ya Lai, "The design of an ontology-based service-oriented architecture framework for traditional chinese medicine healthcare." In *e-Health Networking, Applications and Services (Healthcom), IEEE 14th International Conference*, pp. 353-356, Beijing, 2012.
- [8] Baader, Franz., and W. Nutt, "Basic Description Logics." In *Description Logic Handbook: Theory, Implementation, and Applications*. Baader, Franz. (Eds.), pp. 47-95, USA: The Cambridge University Press, 2007.
- [9] Krötzsch, Markus., F. Simančik, and I. Horrocks, "A Description Logic Primer." In *CoRR*, *abs/1201.4089*, 2012, [Online] Available: <http://arxiv.org/abs/1201.4089> [Accessed 27 April 2016]
- [10] Horrocks, Ian., and U. Sattler, "A Description Logic with Transitive and Inverse Roles and Role Hierarchies." In *Journal of Logic and Computation*, 9(3), pp. 385-410, 1999.
- [11] Baader, Franz., and U. Sattler, "Description Logics with Symbolic Number Restrictions." In *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI-96)*, W. Wahlster, Ed., pp. 283-287, John Wiley & Sons Ltd, 1996.
- [12] Horrocks, Ian., P. F. Patel-Schneider, and D. L. McGuinness, C. A. Welty, "OWL: a Description Logic Based Ontology Language for the Semantic Web." In *Description Logic Handbook: Theory, Implementation, and Applications*. Baader, Franz. (Eds.), pp. 458-486, USA: The Cambridge University Press, 2007.
- [13] Zuo zhihong and Zhou mingtian, "Web Ontology Language OWL and its description logic foundation." In *Parallel and Distributed Computing, Applications and Technologies, PDCAT'2003. Proceedings of the Fourth International Conference*, pp. 157-160, 2003.
- [14] W3 staff, "OWL 2 Web Ontology Language Primer (Second Edition) Publication History - W3C", In *Continuing Education on New Data Standards & Technologies*, 2012,

- [Online] Available: <http://acva2010.cs.drexel.edu/omeka/items/show/5146> [Accessed 20 April 2016]
- [15] W3 staff, “OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)”, *Continuing Education on New Data Standards & Technologies*, 2012, [Online] Available: <https://www.w3.org/2012/pdf/REC-owl2-syntax-20121211.pdf> [Accessed 20 April 2016]
- [16] Pizza ontology [Online] Available: <http://protege.stanford.edu/ontologies/pizza/pizza.owl> [Accessed 20 April]
- [17] Horridge, Matthew., “A Practical Guide to Building OWL Ontologies Using Protégé 4 and CO-ODE Tools.” 1.3 ed. s.l.: The University of Manchester, 2011, [Online] Available: <http://owl.cs.manchester.ac.uk/publications/talks-and-tutorials/protg-owl-tutorial/> [Accessed 20 April 2016]
- [18] D'Aquin, Mathieu., A. Schlicht, H. Stuckenschmidt, and M. Sabou. “Criteria and Evaluation for Ontology Modularization Techniques.” In *Modular Ontologies*, H. Stuckenschmidt, C. Parent, and S. Spaccapietra (Eds.). *Lecture Notes in Computer Science*, vol. 5445, pp. 67-89, Springer-Verlag, 2009.
- [19] Schlicht, Anne., and H. Stuckenschmidt, “A Flexible Partitioning Tool for Large Ontologies.” In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01 (WI-IAT '08)*, pp. 482-488, IEEE Computer Society, 2008.
- [20] PATO tool [Online] Available: <http://web.informatik.uni-mannheim.de/anne/Modularization/pato.html> [Accessed 12 April 2016]
- [21] Noy, Natalya F., and M. A. Musen, “The PROMPT suite: interactive tools for ontology merging and mapping.” In *International Journal of Human-Computer Studies*, 59, 6, pp. 983-1024, 2003.
- [22] Schlicht, Anne., and H. Stuckenschmidt, “Structure-Based Partitioning of Large Ontologies.” In *Modular Ontologies*, Schlicht, Anne., Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra, (Eds.), *Lecture Notes in Computer Science*, vol. 5445, pp. 187-210, Springer-Verlag, 2009.
- [23] Protégé editor [Online] Available: <http://protege.stanford.edu/> [Accessed 12 April 2016]
- [24] OntoGraf [Online] Available: <http://protegewiki.stanford.edu/wiki/OntoGraf> (2014) [Accessed 21 August 2016]
- [25] OWLViz [Online] Available: <http://protegewiki.stanford.edu/wiki/OWLViz> (2014), [Accessed 21 August 2016]
- [26] Ramakrishnan, S., and A. Vijayan, “A study on development of cognitive support features in recent ontology visualization tools.” In *Artif. Intell. Rev.* 41, 4, pp. 595-623, 2014.
- [27] Horridge, M., and S. Bechhofer, “The OWL API: A Java API for OWL Ontologies.” In *Semantic Web 2*, pp. 11-21, 2011.
- [28] Shearer, R., Motik, B., and I. Horrocks, “HermiT: A Highly-efficient OWL Reasoner.” In *CEUR Workshop Proceedings 432*, 2009.

- [29] Lohmann, S., S. Negru, F. Haag, and T. Ertl, "VOWL 2: User-oriented Visualization of Ontologies." In *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8876, pp. 266-81, 2014.
- [30] Micallef, L., and Peter J. Rodgers, "eulerForce: Force-directed layout for Euler diagrams." In *Journal of Visual Languages and Computing*, 25, pp. 924-934, 2014.
- [31] Lohmann, S., S. Negru, F. Haag, and T. Ertl, "Visualizing Ontologies with VOWL." In *Semantic Web* 7(4) pp. 399-419, 2016.
- [32] Lohmann, S., S. Negru, and F. Haag, "VOWL: Visual notation for OWL ontologies." 2014. [Online] Available: <http://purl.org/vowl/> [Accessed 12 April 2016].
- [33] IntelliJ IDE [Online] Available: <https://www.jetbrains.com/idea/> [Accessed 20 August 2016]
- [34] Loeliger, J., "Version Control with Git". California: O'Reilly Media, Inc., 2009.
- [35] GitHub [Online] Available: <https://github.com/> [Accessed 25 August 2016]
- [36] "Spring MVC Framework for Web 2.0." In *International Journal of Engineering Innovations and Research* 1.3, pp. 188-93, 2012.
- [37] Johnson, R., et. al. "Spring Framework Reference Documentation 4.3.2." 2004-2016. [Online] Available: <http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/> [Accessed 25 August 2016]
- [38] Sobernig, S., U. Weiss. Zdun, M., and P. Avgeriou, "Inversion-of-control Layer." In *Pattern Languages of Programs Proceedings of the 15th European Conference*, pp. 1-22, 2010.
- [39] Koopmann, Patrick A., and Renate A. Schmidt, "Forgetting Concept and Role Symbols in ALCH-ontologies." In *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8312, pp. 552-67, 2013.
- [40] NodeJs [Online] Available: <https://nodejs.org/en/> [Accessed 20 August 2016]
- [41] Nginx [Online] Available: <https://www.nginx.com/resources/wiki/> [Accessed 22 August 2016]
- [42] Sarkar, D., and ebrary Academic Complete. "Nginx as a Reverse Proxy." In *Nginx 1 Web Server Implementation Cookbook: Over 100 Recipes to Master Using the Nginx HTTP Server and Reverse Proxy*. Ch. 7, pp. 119-134, Birmingham, U.K.: Packt Open Source Pub., 2011.
- [43] Everett, Gerald D., R. McLeod, and Wiley InterScience. "Testing Strategy" In *Software Testing: Testing across the Entire Software Development Life Cycle*. Ch. 4, pp. 66-78, Piscataway, NJ: Hoboken, N.J.: IEEE; Wiley-Interscience, 2007.
- [44] Parsons, D., and SpringerLink." Unit Testing with Junit." In *Foundational Java Key Elements and Practical Programming*, pp. 219-244, 2012.
- [45] Bibtex.owl ontology developed by Nick Knouf, from Cornell University, nknouf@zeitkunst.org [Online] Available: <http://zeitkunst.org/bibtex/0.1/> [Accessed 03 August 2016]
- [46] BibTeX [Online] Available: <http://www.bibtex.org/> [Accessed 03 August 2016]

- [47] Fenn, J., “Managing citations and your bibliography with BibTeX.” In *The PracTeX Journal* 4. 2006.
- [48] OWLLogicalAxiom in OWL API page [Online] Available: <http://owlapi.sourceforge.net/javadoc/org/semanticweb/owlapi/model/OWLLogicalAxiom.html> [Accessed 03 August 2016]
- [49] Travel.owl developed by Holger Knublauch from Stanford University, [Online] Available: <http://protege.cim3.net/file/pub/ontologies/travel/travel.owl#> [Accessed 03 May 2016]
- [50] Sure, Y., S. Bloehdorn, P. Haase, J. Hartmann, and D. Oberle, “The SWRC ontology – semantic web for research communities”. In *Proceedings of the 12th Portuguese conference on Progress in Artificial Intelligence (EPIA'05)*, Carlos Bento, Amílcar Cardoso, and Gaël Dias (Eds.), pp. 218-231 Springer-Verlag, Berlin, Heidelberg, 2005.
- [51] Vrandečić, D., P. Hitzler, and R. Studer, “DLP - An Introduction”, 2005, [Online] Available: <http://corescholar.libraries.wright.edu/cse/67> [Accessed 10 August 2016]
- [52] Jozefowska, J., A. Lawrynowicz, and T. Lukaszewski, “Frequent Pattern Discovery from OWL DLP Knowledge Bases.” In *Managing Knowledge In A World Of Networks, Proceedings 4248*, pp. 287-302, 2006.
- [53] SWRC Ontology [Online] Available: <http://ontoware.org/swrc/> [Accessed 10 August 2016]
- [54] Bostock, M., V. Ogievetsky, and J. Heer, “D³ Data-Driven Documents.” In *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301-2309, 2011.
- [55] D3 (Data-Driven Documents) [Online] Available: <https://d3js.org/> [Accessed 27 April 2016]
- [56] Keith, J., and J. Sambells, “DOM Scripting: Web Design with Javascript and the Document Object Model” (2nd ed.). Apress, Berkely, CA, USA, 2010.
- [57] Chaomei, C., “SVG and X3D: New XML Technologies for 2D and 3D Visualization.” In *Visualizing the Semantic Web: Xml-Based Internet and Information Visualization*. pp. 124-133, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [58] Batagelj, V., and A. Mrvar, “Pajek - Analysis and Visualization of Large Networks.” In *Graph Drawing 2265*, pp. 477-478, 2002.
- [59] Pajek tool [Online] Available: <http://mrvar.fdv.uni-lj.si/pajek/> [Accessed 27 April 2016]
- [60] Apache Tiles Framework [Online] Available: <https://tiles.apache.org/> [Accessed 20 August]