ONTOMETRY LIBRARY:
A MODERN METHOD FOR THE DISTRIBUTION AND SHARING OF ONTOLOGIES

by

Daniel Kotowski

A Thesis
presented to
The University of Guelph

In partial fulfilment of requirements
for the degree of
Masters of Science
in
Computer Science

Guelph, Ontario, Canada
©Daniel Kotowski, April, 2016
ABSTRACT

ONTOGRAPH LIBRARY:
A MODERN METHOD FOR THE DISTRIBUTION AND SHARING OF
ONTOLOGIES

Daniel Kotowski
University of Guelph, 2016

Advisor:
Professor Deborah Stacey.

The backbone of semantic web technologies is the ontology. This is a powerful structure, which allows for the capture, reasoning and storing of expert knowledge across various domains. Ideally these structures should be developed and implemented by experts in a set domain as well as designed with re-usability in mind. However, often due to the lack of availability and difficulties of discovering ontologies, these structures are repeatedly recreated. Current methods for storing, discovering and sharing ontologies employ similar techniques as to those used for software source code or static web pages. These are exposed to the limitations inherent with keyword-based searches, such as ambiguity with the keywords themselves and therefore, the most relevant ontology may not be discovered. This paper will examine some of the existing techniques used for the storing and sharing of ontologies. It will offer a contrasting method analogous to software libraries to develop a standard to store, share, discover, and distribute common ontologies.
Acknowledgements

Thank you to my family and my wife for your unending encouragement. Thank you to my advisor Professor Stacey for your advice and support. Last but not least thank you to the cats Mumbles and Bandit who kept me company while I worked late at night.
Contents

1 Introduction 1

1.1 Motivation ........................................ 1

1.2 Thesis Statement ................................... 2

1.3 Fundamental Difference Between a Library and Repository ....... 3

1.4 Contributions ...................................... 4

1.5 Document Outline .................................. 5

2 Literature Review 6

2.1 Ontologies: What are they? .......................... 7

2.2 It only seems logical ................................ 8

2.2.1 Ontologies are made of (Parts of an Ontology) ........... 8

2.3 Ontology are designed to be shared .................... 11

2.4 Ontology Repositories ................................ 11

2.4.1 BioPortal ........................................ 13

2.4.2 SOR ............................................. 15

2.4.3 Lightweight Ontology Repository ...................... 16

2.4.4 The Ontology Meta-Ontology ....................... 17
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Wines in relationship to Grapes. Example of how classes and instances relate to one another.</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>“Hierarchical view versus ontological view. (a) Ontologies represent common properties across subclasses, whereas (b) hierarchy reflects a more top-down approach.” Source(<a href="#">21</a> ©2009, IEEE)</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>Ontology Meta-Ontology (OMO) (<a href="#">28</a>)</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>The types of changes different versions of ontologies can go through from <a href="#">25</a>.</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>The adoption layers of the SOL</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>The Meta Ontology used within the Standard Ontology Library (SOL)</td>
<td>32</td>
</tr>
<tr>
<td>3.4</td>
<td>The Meta Ontology has the potential to allow us to have the necessary resolution to see ontologies change over time.</td>
<td>33</td>
</tr>
<tr>
<td>3.5</td>
<td>The overall architecture of the SOL</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>A table describing the general attributes of reusable software from “Measuring Software Reusability” <a href="#">32</a> ©1994 IEEE.</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>A table describing information helpful when reusing software from “Measuring Software Reusability” <a href="#">32</a> ©1994 IEEE.</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>Google search results when using the keyword “date ontology”.</td>
<td>48</td>
</tr>
</tbody>
</table>
A.2 Overview of the the Change class structure of the meta-ontology. . . . . . . 86
A.3 Overview of the the Ontology Domain class structure of the meta-ontology. 86
A.4 Overview of the the Ontology Properties class structure of the meta-ontology. 86
A.5 Overview of the object properties present within the SOL meta-ontology. . 87
A.6 Overview of the data properties present within the SOL meta-ontology. . 88
Chapter 1

Introduction

1.1 Motivation

Ontologies are a relatively new technology, that have various uses in a variety of domains. One technology which heavily relies on ontologies is the semantic web. Currently, the web as we know it, stores data in human readable documents. The semantic web aims to develop technologies that will allow machines to be able to understand the documents and services that they access [34]. This would fundamentally change the way both computers, and humans interact with the web.

Ontologies fit into the semantic web as the means by which to describe the documents, services, etc. Ontologies are machine readable knowledge entities which describe various concepts and objects, as well as the relationships between them. Furthermore, the relationships established can be reasoned upon, which can help aid in any decision making process. In the context of the semantic web, a machine would be able to ascertain the context of a web page (e.g. if a web page is about music, a band, or food). When looking at semantic web services, the ontology would provide context to the functionality of the web service, along with a description of the input expected, and the output of the web service.

Ontologies facilitate complex functionality, but this does not come without cost. The
creation and validation of an ontology is incredibly difficult. Ideally these structures are designed with reusability in mind and intrinsically linked to it via the formal definition. However, currently there does not exist a unified method for doing so. Many research groups/companies who actively use ontologies, are forced to develop their own system/methods to store ontologies. Often as result these ontologies are highly difficult to find, which in turn limits reusability. The lack of usability and accessibility holds ontologies back from adoption, and limits its appeal due to the time/cost investment to have one/many implemented within in a project.

1.2 Thesis Statement

The metaphor of an ontology library is more suited to facilitating the reuse of ontologies over a repository. The metaphor will improve reuse by enhancing discovery through underlying classifications and organizational structures for the ontologies. Furthermore, reuse will be encouraged by adding the notion of ownership and review, and thereby instilling user confidence in the quality of the ontology. Lastly, by adding versioning and temporal change, the user will be able to see what modifications occurred to determine their impact on the ontology. Underpining this is a fundamental shift from viewing ontologies as taxonomies/structured documents but actually looking at them as individual functional unit that have both definition and function.

Standards will be shown by examining existing systems via literature review and against existing software metrics (and ultimately show the lack of them). The library will be designed by examining the existing gaps in what is state-of-the-art for ontology repositories, then by incorporating the lessons learned into the library meta-ontology to fill these gaps.
1.3 Fundamental Difference Between a Library and Repository

At this point I would like to make a differentiation between what currently exists and the metaphor which is being proposed within this thesis. Often when this problem is examined most suggest a technical solution, which often leads to a variant of a source code repository which contains ontologies. The fundamental problem with this approach is that the user is responsible for every aspect of the system. The user must implement the organizational structure of the ontologies, rely on existing search algorithms, while making sure the ontology is of proper quality to be added to the system, as well as so much more. This is an incredibly daunting task.

This should not be surprising, as when organizations set up code repositories they take time and effort to organize the repository in a method that best suits the organization’s needs. Often many newcomers are brought up to speed with the structure and contents of a repository. The technology is extremely general and can be applied to various domains, be it source code, documents, images, etc. Repositories also add the ability to track changes in files over time. This is the extent of the functionality present in a source code repository. Generality is its strength and greatest weakness as, one can add into it whatever you want, while the repository knows very little about what it’s storing.

Though repositories are simple to implement and understand, they may not be the best tool for the job. When looking at software, there are two components: semantic and syntactic; much like ontologies have both semantic and syntactic components [15]. It is clear in the design and function that a repository only concerns itself with the storing and change of syntactic information. The notion of reuse is absent. While in ontologies the semantic information is what users often search for when looking for an ontology as it is the key for its use/reuse. We want to organize the functional “semantic elements” in a logical structure as well as ensure the integrity of each ontology stored within the library. Since repository often heavily rely on crowd sourcing, the integrity/quality of the ontology
usually cannot be guaranteed.

By switching to a library metaphor we are shifting the onus from the user to the system, as well as how we interpret ontologies. In the sense that ontologies will no longer be looked at as files that store structured data (like XML files for instance), but as functional units with purpose and reason.

1.4 Contributions

The goal of this thesis is to increase and enhance the reuse of ontologies. This will be done by considering the metaphor of an ontology library over a repository. Through this endeavour, two key contributions will be developed.

The first contribution is the creation of a framework for the ontology library. It defines several of the key characteristics relative to the ontology definition to allow for classifications and organization by domain. Additionally, the framework defines a strategy to classify changes within an ontology, as well as the severity of the changes, to help a user gauge the impact of these changes. Finally, the framework incorporates the notion of ownership and peer review to ensure the quality of all ontologies present within the library.

To enforce and capture the necessary information for this library methodology, it was necessary to develop a meta-ontology to tie all this information together in a logical manner. This is our second contribution. This ontology leverages the work seen in [28] to develop a rich representation of all the ontologies present within the library. Furthermore, by choosing to use an ontology to represent this meta information, it allows the user to unlock the power of ontology reasoners and apply complex and rich queries to determine complex information about the ontologies present within the library. The details of this ontology can be seen in Appendix A. Lastly, the contributions did not include the development of an interface around the framework, nor did it include seeding the ontology library with an initial set of ontologies. It focused on the design of the meta-ontology and how it would be used within the library framework.
1.5 Document Outline

This section will outline the thesis document. Chapter 2 examines the definition and key components of an ontology while looking at modern ontology repositories’ strengths and weaknesses. This will be followed by Chapter 3 which outlines our approach for ontology reuse and sharing: the ontology library. Furthermore, this section will look at various components of the ontologies while taking into consideration the necessary information to represent each ontology, strategies for managing changes in the ontologies, and finally, defining an overarching meta-ontology to tie all the system components together. Chapter 4 compares the new methodology of a software library against an existing ontology repository, as well as comparing and contrasting our methodology to that of an ontology repository. Lastly, Chapter 5 summarizes the case studies and reveals possible solutions for increasing reuse of ontologies. In Chapter 6 we expand upon our conclusions to extrapolate where potential future work can explore.
Chapter 2

Literature Review

When examining the existing methods for storing, sharing, and the distribution of ontologies, it is important to first examine the definition and structure of an ontology. This will help guide design decisions of the proposed ontology library. It is also important to understand the process in which the ontologies are designed, as it illustrates much of the types of information being captured within an ontology, as well as emphasizing the necessity for reuse. Many of the existing ontology repositories focus their design on domain specific needs, while ignoring many of the fundamentals of an ontology, and thus rely on standard document sharing techniques (such as version controlled repositories similar to subversion or Dropbox). Often these techniques are modified to suit the domain’s needs specifically, and in doing so information from the ontology is lost. Employing these techniques also puts the onus on the user to figure out what they need to find and how to find it.

This literature review will first examine the ontology and its key features, followed by the engineering process of an ontology. With this foundation it is then possible to examine the current offerings of ontology repositories. This chapter will then proceed to look at software libraries and how this metaphor can be applied to the storing of ontologies.


2.1 Ontologies: What are they?

Ontologies are knowledge structures designed to capture and describe objects, processes, and ideas found within our world. These computational artifacts originated from philosophy [17]. Though the use is quite different in the two respective fields, the similarities are also quite apparent. In the context of philosophy, ontologies deal with describing “nature” along with its inherent structure [17]. However, in the domain of machine readable knowledge representation, ontologies are defined as, “a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of phenomena in the world by having identified the relevant concepts of those phenomena. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine readable. Shared reflects that ontology should capture consensual knowledge accepted by the communities” [16] [6]. Though both describe the world, ontologies used within computer science have several constraints so that they can be reused and shared.

Since by definition ontologies represent a shared conceptualization, it would be redundant to perpetually reproduce the same conceptualization. Thus, it is fair to assert that ontologies should be reused. This is further examined in section 2.3.

There are two types of ontologies: transcendent and immanent. Transcendent ontologies are those derived from well defined and accepted knowledge from existing corpora on said topic, while immanent ontologies are based on the content of a domain [21]. An example of an immanent ontology given in [21] is an ontology of all the items in a rail newspaper. The structure of this item would change daily depending on the news of that day. An example of a transcendent ontology would be that of time, as it has an accepted shared definition of what units such as minutes and seconds represent, and this does not change.
2.2 It only seems logical

Ontologies use description logics to define their structure and members. “Description logics (DLs) are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way” [3]. Using description logics to define ontologies, we can view its parts as sets and the relations between parts as an intersection of sets. It is important to note that there is a different set of ontology languages based on “Predicate Logic” though many of these languages were conceived prior to the semantic web (such as CycL and KIF) and were not considered in this work as they are not heavily used in the semantic web space [11][14].

Since ontologies in the semantic web domain use description logics to implement them, it allows for integration with existing reasoners to interpret them [3]. Through the interpretation, we see the functional aspects of an ontology. This exemplifies the distinction between a markup language such as XML over reasoned/interpreted objects such as ontologies.

RDF and RDFS were formal languages used heavily to define ontologies within the semantic web space. However, these languages were limited in the things they could describe. The Web Ontology Working Group of the W3C decided to extend the languages of RDF and RDFS into the Web Ontology Language or OWL for short [2]. The ultimate goal was to create a language that had “a well-defined syntax, well-defined semantics, efficient reasoning support, sufficient expressiveness power, and convenience of expression” [2].

Originally the goal was to simply extend the RDF schema to give added expressiveness. However, the tradeoff between efficiency at reasoning and expressive power created problems when trying to adhere to the original schema [2].

2.2.1 Ontologies are made of (Parts of an Ontology)

An ontology can be quite complex; it is, however, constructed from only a few basic components. These components are common despite the language in which the ontology is represented. The fundamental unit of the ontology is the class. The class should be thought of
as a set classifier, and all the members of this set are defined as *instances* \[21\]. All classes within an ontology are an instance (directly or indirectly) of the class “Thing”. Each of the instances are a subset of a super class, either being “Thing” or a subclass of “Thing”.

Figure 2.1: Wines in relationship to Grapes. Example of how classes and instances relate to one another.

Though these classes are already related amongst themselves via the class hierarchy, they have additional properties that the classes and instances can relate to each other. These relationships are referred to as *object properties* \[21\] \[2\]. Figure 2.1 is an example of object properties between instances of the class wine and the instances of the class grape. With these relations established, we can infer a large amount of information \[21\]. For example we can see what is considered a wine and what is considered a type of grape, along with which wine is composed of which grape.

Classes and relationships can have individuals. Individuals are instances of a class that are real objects. For example if an ontology was describing different models of automobiles, a specific automobile would be an individual. The VIN number for such an automobile would be represented as a data property, within the ontology.
These four parts are atomic structures which can be defined at the time of the ontology creation, and are visible when using any type of ontology visualization software. However, another part of an ontology that is only realized after the ontology is reasoned upon is the axiom. Axioms are rules to be put onto relations defined with object and data properties. When the ontology is then processed by a reasoner, the axiomatic restrictions placed on the ontology will result in both new and potentially changed relations.

Though ontologies have a class hierarchy, they differ from a simple hierarchy. In a hierarchical representation it is difficult to represent food items that have a similar colour, as in a true hierarchal structure oranges, cheddar, and carrots would be represented under fruits, vegetable, and cheese respectively. However, in ontologies since they are based on sets, it is possible to show that these foods are at the intersection of the set of fruits, vegetables, and cheeses [21]. This is seen in Figure 2.2.
2.3 Ontology are designed to be shared

The problem being addressed within this thesis directly relates to the process of ontology engineering. Many of the methodologies and techniques relating to the design and implementation of new ontologies rely on the use and integration of existing ontologies [35][16]. This mitigates the need to recreate components of common ontologies, such as date/time, location, processes, etc. It also allows for the ontology developer to focus on novel parts, without the need to evaluate those existing elements.

Furthermore, [16] methodologies design ontologies with reuse inherently in mind. One of the key criteria discussed is extendibility. [16] states that “An ontology should be designed to anticipate the use of the shared vocabulary. It should offer a conceptual foundation for a range of anticipated tasks, and the representation should be crafted so that one can extend and specialize the ontology monotonically. In other words, one should be able to define new terms for special uses based on existing vocabulary in a way that does not require the regions of the existing definition”. This criteria suggests that ontologies should be designed with reusability in mind as extendability is the foundation of reusability.

Another pillar on which reusability is built on is that of flexibility, which both [16] and [35] address in their development methodology. This is essential when addressing reusability as without flexibility the ontology cannot be reused or repurposed due to the rigidity and specificity in its design.

Therefore when examining ontology design methodologies we can see that when designing an ontology, reusability should be kept in mind (even though it is not directly stated), as reusability requires extensibility and flexibility, and both cannot exist without the other.

2.4 Ontology Repositories

As described in the previous sections, it is difficult to design, engineer, and validate an ontology. Ideally, a user should be able to search for and reuse existing ontologies. However,
currently it is not clear how to locate ontologies and even if an appropriate ontology were found, how the user should reuse it [29]. Smarter, more efficient tools and aids would be necessary for an innovation in ontology technology [9]. This can be achieved through either a library or a repository.

An Ontology library system is an over arching system that oversees groups of ontologies for the purpose of reuse [9]. Though this definition encapsulates a majority of what is required in ontology library systems, there is other information needed to satisfy all the requirements as described above.

Additional aspects which should also be considered when developing such a system, is the notion of trust, quality, and information on who published the ontology. When using ontologies for important life critical tasks such health and medicine it is important to understand that the creator of the ontology was qualified to capture said knowledge. For instance, you would not want a history professional designing an ontology on the intricate interactions between cancer cells, as it would be much more preferable that the ontology was published by an organization or individual who is actively researching cancer.

As these ontologies are published, they may change over time and information about the version of the ontology becomes important. Versioning information, is simply the nature of the change from a previous version. This will help the user understand if the change was an addition, a subtraction, or worse yet a modification of existing relations. This is incredibly important when considering whether or not to update the current ontology being used in an application to a newer version. If the ontology is critical to the systems functionality, it may be wise to hesitate before adopting the newest version of the ontology.

The last and most important aspect to consider when designing any type of ontology distribution system is the user. As described above the user should be made centre in any of these system implementation, as the key goal for any library is for mass adoption; and by targeting the user we will be able to have wider adoption of these types of systems.

The idea of storing and reusing ontologies is not new, and many systems have been proposed. These systems often come in flavours, such as keeping persistent ontologies to
allow for active reasoning or the alternate of keeping all files grouped and letting search
take care of everything. Some examples of systems that allow active reasoning are that of
Minerva[37], Sesame [5], OWLIM[24] and SOR[27]. These systems usually represent the
ontologies as a set of tables and link them via foreign key values, and thus use these rela-
tionships to actively reason and take information out of them; however they require extreme
integration with existing systems and often become very dependent on the individual im-
plementation, thus making them somewhat solution specific. The other flavour is that of
storing them in files and using text search tools or other relations to aid in the location
of a specific ontology. Some of examples of these types of systems are OntoRama[10] ,
Swoogle[8], BioPortal[33][30] and The Lightweight Ontology Library[31]. These ontology
repositories try to store the ontology files and distributed them to users using various meth-
ods. The Lightweight Ontology Library, OntoRama and BioPortal are more traditional
forms of repositories where ontologies are stored and users look through files, while Swoogle
is more of a search engine which indexes ontologies found on the web. This and other unique
systems are examined in section 2.4.5.

As we have seen, there exists many systems that share ontologies. Through this litera-
ture review we will examine in detail BioPortal, SOR, Light Weight Ontology Repository
and the Ontology Meta Ontology as systems representative of the group. Most of the
systems have similar functionality or use the same methodology as purposed in these rep-
resentative systems. Usually these systems are designed with a specific domain or specific
purpose in mind.

2.4.1 BioPortal

One of the larger ontology repositories is the BioPortal project. This project focuses on
the distribution and storage of biomedical ontologies [30]. The project was motivated to
address five distinct challenges with the distribution of ontologies across the World Wide
Web [33]. The challenges addressed by the BioPortal Project as defined by [33] are:

1. Knowledge access and awareness: the realization that there exists several ontolo-
gies in the e-science domain without any central location to find them, as well as no place for ontology developers to stay “up-to-date” with the ontologies that currently exist.

2. **Search:** with the distributed nature of ontologies, there must be a method to search ontologies, which capture certain terms/relations.

3. **Diversity of Knowledge Representation (KR) Languages:** as described in 2.1 there are various languages one can represent an ontology in; the system should strives to accommodate all the major languages in which ontologies are represented.

4. **Overlapping Content:** address the issues of ontology mapping and ontology extension.

5. **Community feedback:** currently when ontologies are stored they are often located on a website without any feedback from users of the ontology, the system will try to address this by allowing for users to rate and leave feedback on each individual ontology.

These are the existing challenges with developing an ontology repository [9]; however, without context of the average users needs, it will be difficult for these challenges to be addressed directly. Though BioPortal may be a technical marvel it may not necessarily address the issues active users of ontologies need resolved. Without directly addressing these issues, the tool may be yet another barrier of entry for potential ontology users. Nonetheless, BioPortal did preform usability testing, along with taking into consideration feedback presented on user postings on the projects website [33]. The specifics of this testing and feedback have yet to be disclosed.

BioPortal is implemented as a web portal where users can log on, upload, rate, comment, edit, and map ontologies within the system. The key contributors to the ontologies present within this project is the biomedical community, although it is not restricted to this community. Since BioPortal is an open community there are no restriction as to who can contribute to this project [30]. There is no method for screening ontologies before they are
posted, as the community is expected to perform peer reviews of the ontologies within the system, and ultimately crowd source the quality of the ontologies.

Since BioPortal heavily relies on the user community to guarantee quality, it may be fair to assert that industry stake holders may be reluctant to trust ontologies present in this system. The thesis hopes to assert that there are more reliable means to ensure quality as discussed in Chapter 4.

This project used methodologies similar to those used in source code repositories. The difference between an ontology and source code is that while the ontology is represented in a file similar to source code, they have an inherent functionality to them that must be taken into consideration whenever developing any means to store and distribute ontologies.

### 2.4.2 SOR

The next repository system to examine is the Scalable Ontology Repository (SOR). The goal of this system is to allow users to search for ontologies as well as reason against the ontologies which are presently in the system [27]. The ontologies within this system are stored within a relational database. Each class is represented as a table and the relations between the classes are the relations between each table. Thus in essence, the database schema represent the ontologies [27]. Storing the ontologies in this manner causes them to be in a persistent state, which allows active reasoning against all the ontologies which are stored in the system. SOR allows its users to use the SPARQL language to query the ontologies within the system [27].

The primary focus of this project was to be able to reason across multiple, and very large ontologies, actively though this is a powerful tool if your system requires many ontologies to run. However it does have basic search. The faceted search used by SOR returns information such as the URI, textual descriptions, the list of classes, subject relations and object relations [27]; however SOR extends this is by allowing the user to explore via relations to other classes and self guide the search results.
This system is uniquely designed as it repurposes the database to fit the paradigm of the ontology representation. This method strictly focuses on the technology of how to represent and share but does not look at quality and trust. Rather it assumes that the user will be using the ontologies through SOR’s system as the ontologies are stored within the system.

2.4.3 Lightweight Ontology Repository

The following section is excerpted from the paper [25], which was written by myself and Dr. Stacey, and presented at the KEOD 2012 conference.

“The Lightweight Ontology Repository enables the sharing of ontologies between multi-agent systems [31], it was design so that ontology designers and agents can use this system with common web standards to publish and retrieve ontologies, along with associated metadata. The agents could access ontologies via a REST style web service. With this project we see more emphasis put onto extra data associated with the ontology. Much like the SOR this repository keeps track of additional data via each ontologies metadata.

The metadata stored within this system is the same as the source Description Frame Work Schema. This ontology keeps track of information such as the ontology name, language, version and brief description, as well as information about the creator and how to contact them [31]. This additional data gives a more complete understanding of the ontology begin stored within the system, however it does not give a complete view. It is an abstracted view, and further processing will need to be done to understand what the ontology is actually describing. The agents would need to have a pre-existing knowledge of which ontology they need, and can validate against the metadata, to confirm they received the correct ontology.

Though this system was designed with a specific domain in mind, it did emphasize the need to store and keep track of each ontology’s metadata. However, this system lacked search functionality, due to the fact that the only way to receive ontologies was through request of exact ontology names. There is no notion of trust or quality within this system;
the users or agent must assume that all ontologies stored within the system are valid, and
the information they are describing is accurate.”

2.4.4 The Ontology Meta-Ontology

The Ontology Meta-Ontology (OMO) is not an ontology repository but a means to describe
various ontologies. Originally designed for the KAON project [4], the ontology is designed
to be part of an infrastructure to help reuse, evolve and distribute ontologies [28]. The
ontology was designed to capture meta data about existing ontologies and allow a reasoner
to search across it. This ontology however was applied only to individual ontologies.

![Figure 2.3: Ontology Meta-Ontology (OMO)](image)

The ontology is described in Figure 2.3. Some of the meta information described by
this ontology are things such as key terms or concepts of the ontology, who authored the
ontology (as well as their credentials), and where it has been used. Though the OMO
does describe several features of any ontology it does miss certain features which prevent
it from being truly general. One of these is domain; the ontology being stored could be a
member of any number of domains but this is not described within the OMO, nor does it
describe the relations between different domains. Another feature which is missing is the
notion of change and versioning. This is important to an ontology as a community’s or
individual’s modifications can have many implications on what the ontology represents and
how to reason with it (this is described in detail in section 3.3). Another feature missing from the OMO is the notion of intellectual rights: is the ontology open source? Can it be modified or distributed freely? These questions should be captured by this ontology.

Though the OMO is missing some features it does include some powerful information for the end user such as where this ontology has been used before, which party/organization originally created this ontology, where has it been applied, as well as categorization of several terms [28].

The notion of an OMO is sound as it helps categorize and describe what an ontology is trying to represent. However, it needs to be expanded to take into account multiple ontologies, versioning, intellectual property, and domain specificity. This methodology for capturing meta-information of digital objects can also be seen in other domains. For example, this Dublin Core Ontology [1] is designed to capture digital contents meta-information.

2.4.5 Non Repository Repository

We have examined several systems thus far that were conceived specifically for ontology storage and reuse, however there exists a set of systems which are specifically designed to search for ontologies with the intention of reuse. One such system is Swoogle[8].

The Swoogle project is a web crawler that searches through and indexes semantic web documents [8]. Its primary foci are to aid the user in finding appropriate ontologies, instance data associated with those ontologies as well as characterizing the semantic web [8]. It uses Google web services to scout out documents that contain the proper ontology format and then indexes them, scans them, and tries to capture relevant data. This system approaches the problem of reuse by trying to capture what already exists and puts the onus on the user to figure out how to reuse and interpret the ontologies that the search returns. Although this system removes the need to actually store the ontologies, it still runs into the issues we have seen with BioPortol in section 2.4.1: trust and quality, user knowledge, inaccurate or difficult to gauge results, learning and interpreting the ontology.
There are other search engines similar to Swoogle, such as the Ontology Look Up Service [7]. This service provides a search interface to that of Swoogle, however, it limits itself to ontologies specifically related to the biomedical field, similar to that of BioPortal. Furthermore, though the interaction interface is similar to Swoogle, its implementation is more inline with the repository model employed by BioPortal.

2.5 Analogous Methods for Storing Ontologies

The notion of reuse has been prevalent throughout all of computer science. It is most notably seen in software engineering. Early in computer science one problem/challenge was that of reusing software artifacts to save time and costs in the development life cycle [26]. Often various pieces of software would be developed that needed to have similar functionality as other pieces and instead of re-writing it the notion of a software library was presented [26]. This high reusability allowed for various programming languages to expand and gain wide acceptance, as the users did not need to reinvent commonly used structures that were used by many.

The strategy employed by software engineering is that of software libraries. Software libraries are artifacts which are commonly used to structure code with similar functions, abstracted so that they can have a generalized purpose and not be intrinsically dependent on the software being developed [26]. This notion of abstraction can be seen in ontology engineering as purposed by T. Gruber in [16]. The parallels between ontologies and software do not finish there, as another such property that software and ontologies share is that of common purpose; software is designed with a function or task to accomplish, much the same as ontologies. Often ontologies are viewed as strictly hierarchies or taxonomies but this is not the case as we can see in [21], [16], and [3]; we see that ontologies are engineered with a specific purpose in mind, that can be validated and tested, as well as reasoned upon, much like software, thus it would seem that the modality of the software library may fit the problem of reusing ontologies.
Software libraries are designed around distinct criteria which include: abstraction, selection, specialization, and integration [26][18]. “Abstraction plays a central and often times limiting role in each of the other facets of software reuse” [26]. Specialization focuses on each component of the software that serves an individual and specific purpose and does not attempt to do everything. Selection refers to the ease of location as well as the clarity of what each artifact does, so that the user can easily decide if the artifact is the one necessary for their purpose. The last facet, integration, refers to the interface by which to interact with the artifact; this is especially important when trying to integrate this with a piece of software. This would also be important when looking at ontologies as often ontologies must be pulled into work in conjunction with each other to describe a larger or more complex concept.

Other features that are often captured by a software library are things such as a graphical user interfaces, a standard framework which includes documentation and legal restrictions on the library, a method to describe the domain in which it belongs (business, science, etc), and lastly and most obviously documentation to support the library [18]. Though these features may be described as meta-features, we see these as the features that need to be captured for ontology reuse and sharing.
Chapter 3

Proposed Ontology Library

This thesis will examine characteristics required to establish a methodology to facilitate the sharing and reuse of ontologies. In Chapter 2 we examined a variety of characteristics that are required to be captured, accessed and presented to the user, as well as facilities required for reuse when attempting to share ontologies. Additionally, we examined the fundamentals of what ontologies are, the components of their constitution, and the engineering principles used to construct them. With all this in mind, we will develop a framework to establish the feasibility of such a library.

In this chapter we will describe a new framework for the exploration, sharing, and distribution of ontologies to facilitate ontology reuse. It will focus on lessons learned in Chapter 2 through choices, and design principles, we will take advantage of the notion of software libraries and apply it to the domain of ontologies, in order to increase reusability. It will also look at social aspects of the system which cannot necessarily be designed technically but must be examined, as ultimately the community must be responsible for the ontologies present, to parallel how software communities are responsible for software libraries.
3.1 The Standard Ontology Library

Ultimately the Standard Ontology Library (SOL) is a simple framework which can aggregate ontologies in a logical manner increasing the structure and organization of the ontologies present within the framework. This will result in greater reusability \[22\]. In addition, several design principles were considered when designing the SOL. We are establishing the SOL as a framework so that we can focus on the design of the framework and not be bogged down by challenges of implementation.

The SOL differentiates itself from the techniques seen in Chapter 2 by altering the perspective by which we examine each ontology. Many of the systems examined in Chapter 2 derive their framework by viewing ontologies as searchable text files. This leads to them being stored haphazardly and without structure; and then the systems just uses keyword based searching to find them. Consequently, this is not ideal, as ontologies are not simple structured data files, but are rich representation of knowledge, that when leveraged in tandem with a reasoner are able to infer and reason against the domain knowledge that they capture. Hence, it would be fair to consider ontologies as functional units, as opposed to static data representations. For this system we are using the metaphor of a software library, where each ontology serves a purpose, and like purposes are grouped. Each ontology would be segregated by domain (purposes) which allows a natural segregation opposed to the often “lump them all together” seen in systems such as BioPortal \[30\]. To help facility this grouping the SOL ties the ontologies together via meta information represented by a meta ontology. This presents two distinct benefits, as it is capturing all the information required to determine if the ontology fits the users’ needs. As this information is stored in an ontology, it allows us to facilitate reasoning against this ontology, which results in the ability to determine additional information through these queries. This strategy is more conducive to searching through ontologies over traditional keyword based searching.
3.2 Meta Information and Domain Categorization

Much like a book’s title does not always describe its subject matter, an ontology’s title or contents may not fully indicate the domain that the ontology covers. Some ontology languages have addressed parts of this issue with description tags at the head of each file, where the ontology creator may choose to describe the ontology, some of the important components, and include annotation for certain components, such as deprecation for specific classes and relations, and version numbering \[2\]. Some information which is captured is:

- Name
- Author
- Version number
- Description
- Depreciation of a Class or Relationship

However, some information is not captured within these description tags. Elements which are not captured are domain, identification of temporal changes between versions, who reviewed the ontology and how has it been verified and validated. These are all important elements when considering an ontology for use. Furthermore this meta information can describe the foundation of any ontology. Often this information is left unstructured or undefined, and some sort of keyword search is employed in hopes of either finding this information or the associated ontology. Contrary to this philosophy, ontologies themselves are a fantastic vehicle for structuring and querying information and knowledge. Why not structure the meta information of many ontologies within an ontology?

When examining an ontology, we can tell from its definition that it is describing concepts and aspects from the physical world. However, they do not describe much about themselves. As we see in Section 3.3, versioning plays an important role. However, ontologies have several aspects which need to be captured, so that a user can decide if they should use it.
For example, the questions following must be answered before use: what is the ontology describing? What field is this ontology from? Is this ontology old? Is this ontology new? Has it been tested? Who published it? plus many, many more. It is important that these questions are answered. One method that works well is using meta data. This is where we can describe the meta data as an ontology as it is a structure and also has extra information. We can query it like a regular ontology. An example of this can be seen in the OMO meta ontology [28]. However, we will extend the definition within our system to capture other data relative to the needs of a library.

### 3.3 Version Control

Many systems seen in Chapter 2 do not consider how an ontology changes over time. This is an important factor to capture, as ontologies will continue to be reviewed and changed as the knowledge of the domain becomes more clear, or evolves. The systems reviewed do take change into consideration, by simply adding a version number to the ontology. Though a version number indicates a change, it does not indicate the change in the ontological structure or meaning. This could negatively impact the user as the changes may add unwanted effects to the knowledge base of the application. However, these changes may also be required to fix errors in ontological representation. This makes it important as such changes are conveyed to the user to allow for accurate assessment, of changes and the resulting effect on the application.

We came to this solution based on deciding on fundamental changes that can be applied to an ontology. Thus, for the purpose of this thesis we are looking at changes at the class(set) and relation(intersection) level. Changes will be addressed in this system by breaking up the changes into three distinct categories:

**Additions:** when new concepts and associated relations are added to the ontology.

**Subtractions:** when concepts or relations are removed from the structure of the ontology.
**Modifications**: when the structure/relations of the classes are reorganized within the ontology.

These changes can have varying degrees of impact to the end users. For example if the ontology has classes or relationships added to it, the resultant additions should not affect the existing relationships. Since the relations within ontologies are set intersections, and since set intersections are associative, addition of the new relations does not affect the existing set intersection.

\[ A \cap (B \cap C) = (A \cap B) \cap C \]

When considering the removal of a class or relation, we are removing an intersection of that set and what we can see is that an ontology that states:

\[ A \cap (B \cap C) \]

By removing C (either the set or intersection) we are left with:

\[ A \cap B \]

Through this deletion we lose the relationship between B and C and A and C, but maintain the relation between A and B. This is more detrimental to the user than a simple addition, however less so than the structural change seen in the modification.

The modification fundamentally changes the structure of the ontology from:

\[ A \cap (B \cap C) \]

to a relation of

\[ A \cap (B \cup C) \neq (A \cap B) \cap C \]

We can see a visual representation in Figure 3.1. It is easy to see that an addition does not affect any of the fundamental relations of the existing ontology. However, the deletion of classes and relations are more detrimental, as these relations no longer exists in the knowledge representation. Modifications to the relations change the shape and appearance of the ontology. During a modification, the relations between classes, and representation of classes and elements may be changed.

Now that we have established the types of changes, and their associated impact on
Figure 3.1: The types of changes different versions of ontologies can go through from [25].

ontologies, we can now establish the strategy to represent and handle these changes within
the SOL. Obviously, it will require a versioning strategy. Numbering strategies can be
employed for quick identification. It will be a three number code involving the following:

[Major Version].[Minor Version].[Sub Minor Version]

The Major Version number is the current version of the SOL. This is applied to all
ontologies as the SOL grows. For example, if the SOL contains Date and Math ontologies
in Major Version 1, adding a Location ontology to the library would move the version to 2.
The second number, Minor Version defines modifications to the ontology. For example,
if any relations change this version will be incremented. Finally the Sub Minor Version
reflects additions or subtractions. Even numbers denote an addition to the ontology, while
subtractions are denoted by an odd number. For example if an addition and a subtraction
occurred incrementation would be to the next Sub Minor odd number. If modification
and subtraction were to occur the **Minor Version** number would increase by one. This allows an ordered quick guide to see what kind of changes have occurred.

Additionally, the SOL will capture what actually happened. This will be done via the meta ontology. The meta ontology describes the relations that changed via properties and descriptions. An example of this can be seen in Figure 3.3.

It is important to note this is a rudimentary method for capturing change within the library, and can be expanded further. This is explained in Chapter 6: Future Work.

### 3.4 Quality Control

When examining any engineering process, one of the most important steps is to validate what was created accurately accomplishes the task that the item was designed for. This is true for ontology engineering [16]. With the ontology library, validation and quality are just as important. To adopt this platform and guarantee the reuse of the ontologies present within the SOL, we need to guarantee the quality of the ontologies being stored. If each ontology is quality then the quality of the overall library increases.

Many of the systems examined in Chapter 2 use crowd sourcing for quality control. However this gives the “use at your own risk” feeling as there is no way to guarantee users honesty or truthfulness in rating. It makes it difficult for users to assess if an ontology is adequate for a mission critical application.

The SOL uses a more proactive approach to ensure quality. To ensure this quality a two prong approach is employed. The first is a layers adoption model where after the ontologies are proven of quality, and generality is established, the ontology will eventually migrate through the layers. The second prong is review by a committee of domain and ontology experts.
3.4.1 Layered Adoption

Ontologies are often placed into repositories/stores without review. Many of the systems reviewed in Chapter 2, especially BioPortal [30], tend to allow for an adoption methodology, adding it in and let it expire when the ontology is not used or lacks quality. This causes a bulk surplus of ontologies within the library which cause it to be over encumbered and difficult to search through. To use such a system, users would be required to look through and analyze each ontology for its fitness related to the problem they are attempting to solve, or domain knowledge they are trying to access.

The SOL drives an adoption approach similar to that of the UNIX security model/Ring security model [23]. This model entails as, the closer you move to the kernel the more privileged the user is required to be. This is similar to the model presented in the SOL. As ontologies are reviewed by committee those of higher use and quality will be elevated in the library. The adoption model can be see in Figure 3.2

The outer most layer is an experimental space where test ontologies can be deployed. They are not reviewed and should be experimented with like test software not necessarily industry grade but functional.

The following layer is industry domain specific. These ontologies will be divided by domain and strongly tied to a specific industry, or field of study. Examples of such ontologies could include tax ontologies which define rules and regulations for various jurisdictions. Another example could be a medication ontology for the pharmaceutical industry which describes components of common medications, etc.

The subsequent layer are industry ontologies which are domain unspecific, i.e. shared by many industries. Some types of these ontologies could included a variety of things such as jurisdictions.

The centre layer is the core. These ontologies are generic and are used by all regardless of the industry. For example this could include:

- Date/Time
Figure 3.2: The adoption layers of the SOL

- SI Units
- Physical Constants
- Categorization/Classifications
- Language descriptions

Migration via layers would be dependent on adoption and shared use. This implies only the most commonly used ontologies of the highest quality would end up in the core, while
other may only remain in the periphery, either industry or industry shared.

Having this kind of model helps elevate the ontology saturation effect and changes the subtractive model implemented by the other systems to that of an additive one. This allows the user to only see relevant and quality ontologies when deciding which ontology to use as well as increases the reusability of them.

3.4.2 Role of the Committee

Earlier in this chapter, we made the connection between looking at ontologies as a functional structure as opposed to that of a taxonomy. Due to this paradigm shift, we can leverage various techniques and principles employed by software engineering. Specifically, when examining the quality and membership of the ontologies present within the SOL. As we have seen in Chapter 2, many of the systems do not concern themselves with the quality of ontologies present and rely on the user to judge quality and reusability through open commons. In contrast, the SOL is designed with the notion of ensuring quality. Much like in software languages, a committee oversees the specification and the contents of the library.

Having a committee in place will allow members to review: the ontologies, domain coverage (both syntactical and semantic correctness), proper categorization, and when to promote or demote an ontology to the correct layer. It is important to note that this does not imply that there will be a single committee governing the entirety of the ontology or even the entirety of single layer, but there will be multiple committees governing various domain specific ontologies within each layer. These committees will then work in concert to publish the next version of the ontology library.

Many of the methodologies reviewed in Chapter 2 relied heavily on group user input for review of ontologies. However, this leads to several assumptions when a users chooses to use an ontology within their project, resulting in a state of uncertainty about the quality of the ontology. There is no guarantee that the user will be using standard test and refinement methodologies to develop their ontologies. Reviews posted may be skewed based on the
individual uses or ratings done by the creator of the ontology. Furthermore, ontologies may exist which have not been reviewed, or even examined by another person existing within an open commons based repository. Finally, we have no reassurance of the expertise of the ontology publisher in that domain. This model has worked well in various open source projects as seen in the Apache foundation, the Linux Kernel, and the Java language. All these projects have community involvement in development while items are thoroughly reviewed before being added to a "release" of any project.

By having a committee based review methodology, we are now shifting the insurance of quality, and correctness from the user, who wishes to use the ontology, to the creators and maintainers of the library and the ontologies within it. This committee can accept or reject ontologies based on various factor including domain, correctness, organization, etc. Furthermore, if the committees are broken up by domain, they would have sufficient knowledge of the domain to understand if there are missing concepts or if ontologies require additional information. Finally, they can ensure completeness of the information present within the meta ontology along with updating the meta ontology if any changes to the organization occur while maintaining an ontology within the library.

This committee will become the gatekeeper to ensure all ontologies within the SOL are of high quality, properly maintained, and well organized via the meta ontology. Through this we shift the responsibility of ensuring quality from the user of the ontology to the creators and maintainers of the ontology.

3.5 Meta Ontology

At its core, the standard ontology library is driven by an ontology that is used to annotate other ontologies. Annotation of an ontology is not a new concept. Many languages have facilities for the user to annotate various portions of an ontology. For example, the OWL language has facilities that enables a user to annotate relations and classes with some meta information. However, this is done per ontology. To access this information would require
Figure 3.3: The Meta Ontology used within the Standard Ontology Library (SOL)

each ontology to be loaded before it can be analyzed and reasoned against. This would have to occur before the decision of determining whether or not the ontology is suitable for use. Alternate to this approach is to extract this information and have it represented via a
The meta ontology is designed to foster reuse by specifically hoping to increase discovery and by developing a classification structure to organize the ontology in various domains. Furthermore, we want to increase stability and confidence within the ontologies present within the library by encompassing ownership of the individual ontologies. Finally we want to represent ontological change over time via capturing the change sets within an ontology. This will allow us to develop a picture of the information via reasoning to create a temporal map of the ontology over time.

Figure 3.4: The Meta Ontology has the potential to allow us to have the necessary resolution to see ontologies change over time.

With a meta ontology we can have an overarching ontology to represent pertinent information for all the existing ontologies within the library. Specifically, we can represent important information such as dependencies, author, versioning, categorization, etc. Furthermore, we can apply the information that may not be currently captured such as types of changes which occur in the ontology, the current categorization of the ontology, and which ontologies does this ontology depend on or ontologies that are dependent on it. This information is key for when a user is discovering an ontology for use in their project. Since the motivation behind the SOL is to encourage reuse of existing ontologies, it requires a mechanism to categorize the ontologies within the library. At its core, the SOL requires some way to store and manage information on various ontologies from disparate and distinct
domains. For this task the SOL uses a meta ontology to capture key information about each ontology stored within the system. Inspiration for this ontology was taken from the ideas presented in the OMO [28]. However for the purpose of the SOL, it was necessary to define our own ontology so that it could better capture the structure of the library, the delineation between layers, a methodology to characterize change and versioning, and the organizational structures of ownership over the ontologies.

When designing the meta ontology, we sought to capture fundamental concepts with regards to domain classification of ontologies, a notion of temporal change of an ontology over time, and the concept of ownership. To begin with, the domain classification was defined via the class hierarchy of Ontology Domain. This class has three separate subclasses that include Core, Industry and Industry Specific. This allowed us to model the layers adoption approach described in Section 3.4.1 Furthermore these high level classifications can be further subclassed into sub domains under the parent domain. For example, the Core domain can have a subclass such as “SI Units” where ontologies related to describing SI Units, or the Industry Specific domain can have subclass “Banking” where all ontologies related to banking or banking process can be described under this domain. These classes can be seen in Figure 3.3.

To organize and capture temporal changes, we used both the Version and Change class hierarchy and the relations between them. The Version is directly linked to the Ontology class via an inverse relation “hasVersion”. This allows an instance of an ontology to have many different versions, as well as through this association an ontology’s version can then be linked to the Change class via the Version class. This is done through there relation “hasChanges”. This allows us to associate a change set relative to any ontology version. Furthermore, the Change class has three subclasses describing the three types of changes defined in section 3.3: Addition, Subtraction, and Modification. With this information being captured by the ontology we can now reason against it to developed a temporal timeline of an ontology as seen in Figure 3.4. This would allow a user to understand how any change would impact the functionality of the ontology, and how this would impact the
overall system it is a part of. The final concept captured by the meta ontology related to versioning is that of depreciation. However, it is not directly related to version but related to the ownership of the ontology.

Capturing ownership with this ontology differs from the standard approaches seen both in the OWL language and the OMO [28]. These approaches tie the ontology to a specific author, whereas the meta ontology ties a specific ontology to an organization. This draws parallels to that of software libraries, for example, the Java standard IO library which is not authored by a single person but maintained by the Sun Microsystems. Furthermore, when Sun Microsystems was purchased by Oracle, the ownership and maintenance of the library moved onto Oracle. To model this the inverse reaction “isOwnerOf” has been established between the classes *Ontology* and *Organization*. To further enrich the ownership definition the class *Organization* has a relation between itself and the class “*Person*”. Subsequently, the class *Person* has a relation “hasResponsibility” to define the role of the person in relation to an organization. This will be useful when considering whom to contact about changes within an ontology or if you have questions about the ontology. Ownership can help a user determine who is actively maintaining the ontology or if the ontology has become abandoned. When an ontology no longer has an owner organization associated with it, it becomes deprecated. This allows the user to understand that no one is maintaining this ontology and serves as a warning. Additionally if another ontology maintained by another organization heavily depends on this ontology, it can take ownership of the ontology and thus remove the deprecated status from the ontology.

Ownership of an ontology can also prescribe intellectual property and licensing of the distribution of the ontology. This is captured via the class *IP*.

Finally information which many systems already capture is still useful in the context of the SOL. Thus the meta ontology caputres this with the *Ontology Properties* class. This class has the sub classes *Description*, *Statistics*, *Syntax Language*, *Language*, as well as class *Deprecated*. The class *Description* is specifically for the description of the ontology as defined by the owners of the ontology; this allows them to have a textual summary of the
domain that the ontology is attempting to capture. The Statistics class is used to capture empirical information about each ontology. Information captured here would include the number of classes, the number of relations, etc. This information is useful when attempting to gauge the size of an ontology. A user may gauge whether or not they wish to use the ontology or there may be another in the same domain or sub domain that is smaller and or more specific to their needs. The class Syntax Language describes the syntactical language or languages in which an ontology is defined (e.g OWL). The Language class describes the different natural languages that the ontology is described in (e.g English, Polish, German). Finally the class Deprecated relates to whether or not an ontology is deprecated. This information is connected to Ontology through the relationship “hasProperties” between the class Ontology and the class Ontology Properties.

As described, this ontology captures much of the information necessary to help describe an ontology and its most important properties. Furthermore, it allows us to use a ontological reasoner to establish additional information that can further enrich our view of the ontology. Finally, this ontology has inherent flexibility that allows for further extension for additional use cases that are outside the scope of this thesis, but are outlined in Chapter 6 Future Work. Additional relations and the full Meta Ontology can be seen in Appendix A.

3.6 System Architecture

In this section, we would like to describe how the meta ontology and the library would work in a potential system architecture scenario as this will help illustrate the flexibility of this solution. As we are grouping the ontologies in a method similar to that of a software library, we would like to make access to this as flexible and streamlined as possible. Many of the systems and frameworks examined in Chapter 2 were designed specifically for one domain, or to be used as a component of a specialized system [27]. To promote reusability the SOL must be as flexible as possible to allow for potential integration with any system. With this in mind a Representational State Transfer (REST) API based interface is used
to query and access the stored ontologies. This protocol is based on the HTTP protocol [20]. HTTP is the standard protocol by which software accesses web content. This kind of simple protocol allows for access to the SOL by a variety of technologies. Additionally, having this library stored externally allows for “on demand access” to the latest versions of the ontologies, and can be distributed similar to modern package manager systems such as CPAN [36].

Figure 3.5: The overall architecture of the SOL

Behind the interface layer exists a reasoner which will translate the request, into a search against the meta ontologies to aid in accessing only relevant ontologies. There are several reasoners available to be used with ontologies and any of the common ones can be placed within this framework. However, all requests will not be required to be reasoned against as direct access for a specific ontology is also possible, as direct resource access is possible via the API [20]. However, at this point the meta ontology can be leveraged via requests.
to identify and dependencies required by the ontology. For example, if an ontology defining schedules is requested, it will depend on additional ontologies describing date, another describing time, and a third describing location. Through the definition described within the meta ontology, the system could facilitate the additional transfer of the dependent ontologies.

The final portion of this system is the store of the actual ontologies. Due to the flexibility of this design this can be either centralized or alternatively a decentralized model is possible. As ontologies were designed with the idea of the URI, the maintainer of each ontology could potentially host each ontology and the library could facilitate the link between the resource and the requester.

As mentioned earlier, the goal of this system was to remain as generic as possible. A high level architectural diagram can be seen in Figure 3.5; it can be seen that each component can be interchanged as well as independent of each other. Though there is one caveat that depending on the reasoner, the meta ontology may be interpreted differently, and thus query results may vary. However this will be outlined further in the assumptions section of this chapter (Section 3.7).

### 3.7 Assumptions

When designing this methodology, a few assumptions were taken into consideration while designing the SOL. The first assumption is that a set of ontologies exist to populate the library. Currently, due to the lack of organization and methods for finding ontologies in a semi-central location, it may be difficult to establish an initial population of ontologies within the library. The second assumption is the willingness of individuals and domain experts to be present on the committee portion of the ontologies. The third assumption is that the authors and maintainers of the ontology present within the library are willing to make changes and conform their ontologies to the SOL library standards. Lastly, the final assumption is that the meta ontology will be translated into other languages so that it can
be consistent across all reasoners used to interpret the meta ontology. This would allow for consistent results when querying the meta ontology.

3.8 Summary

At this point it is important to summarize what the SOL is as well as to reiterate key design features established within this chapter. The primary purpose is to facilitate the sharing and reuse of ontologies. Recognizing that ontologies are actually functional structures and not taxonomies, it became apparent that a Library would be a much more suitable metaphor than a repository for promoting ontology sharing and reuse.

This chapter described key aspects of reuse relative to ontology domain classification; specifically, when attempting to classify an ontology within a domain it is important to classify various pieces of meta information. Furthermore this chapter investigated and developed an understanding of how an ontology changes over time. Lastly the SOL takes the notion of committee review to ensure that the ontologies are accurate and represent what they are supposed to. Subsequently, proposed methodologies to help ensure quality and constancy of the ontologies present within the SOL, specifically this chapter demonstrated how the SOL employs an acceptance model to allow for gradual adoption.

This chapter analyzed and constructed key methodologies for dealing with the shortfalls identified within the literature review including identifying the necessary information required for ontology discovery, etc.

Finally this chapter developed an ontology to serve as the core method by which to organize all ontologies within the library.

This shifts the onus from the user of the ontology to validate to the creators and maintainers of the ontology and its library. This thesis wishes to establish a framework to help facilitate the active reuse of ontologies. The SOL has been designed specifically to do so. It captures a variety of meta information to help determine the ontology’s roles and what it is describing. The SOL organizes each ontology by domain to allow for logical groupings by
purpose opposed to relying on search for discovery. It considers change over time to allow users to have the proper information to know how it will effect the current implementation of the ontology.

In the next chapter we will validate this approach as a viable option to improve the reuse of ontologies. This will be done through various case studies.
Chapter 4

Case Study

In this chapter we will look at two case studies to compare and contrast the metaphors of an ontology library with an ontology repository. First, we will establish context for the case studies. We will then outline the goal of each case study, as well as key factors and actors. We will conclude with a detailed analysis of each case study relating to the overall goals of this study.

4.1 Background and Purpose

This thesis set out to examine whether the metaphor of an ontology library was more suitable to help promote the sharing and reuse of ontologies. Specifically, this thesis sought to create a new generic library framework to allow for the storing and sharing of ontologies, to improve reuse by allowing for structuring and classification of the domain ontology. Moreover, the library framework will improve the overall quality of the ontologies being stored, and thus instil confidence in the users of the ontologies from this library. Finally, this thesis will improve the way we track changes in ontologies so that a user can get an understanding of what changes occurred and how they impact the ontology. This will help with the reuse of future versions of an ontology.
Therefore, the purpose of these case studies is to illustrate the benefits of using the metaphor of an ontology library over a repository. Furthermore, the case studies will highlight and compare the differences when attempting to discover ontologies for reuse. Specifically, the case studies will consider the following metrics for reusability.

### 4.1.1 Reusability Metrics

To help facilitate analysis through these case studies, it is important to establish metrics to identify key factors to compare each ontology storage method against. In Chapters 2 & 3 we have established that ontologies are not simply searchable taxonomies but functional structures. We can draw parallels from software engineering metrics and map them over to the ontology space. We can look at attributes and information that improve the reuse of software and examine how these attributes are represented within the disparate methodologies being examined. These attributes and information for improving software reuse can be seen in Figures 4.1 & 4.2.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of understanding</td>
<td>The component has thorough documentation, including self-documenting code and in-line comments.</td>
</tr>
<tr>
<td>Functional completeness</td>
<td>The component has all the required operations for the current requirement and any reasonable future requirements.</td>
</tr>
<tr>
<td>Reliability</td>
<td>The component consistently performs the advertised function without error and passes repeated tests across various hardware and operating systems.</td>
</tr>
<tr>
<td>Good error and exception handling</td>
<td>The component isolates, documents, and handles errors consistently. It also provides a variety of options for error response.</td>
</tr>
<tr>
<td>Information hiding</td>
<td>The component hides implementation details from the user, for example, internal variables and their representation. It also clearly defines the interfaces to other operations and data.</td>
</tr>
<tr>
<td>High cohesion and low coupling</td>
<td>The component does a specific, isolated function with minimal external dependencies.</td>
</tr>
<tr>
<td>Portability</td>
<td>The component does not depend on unique hardware or operating system services.</td>
</tr>
</tbody>
</table>

Figure 4.1: A table describing the general attributes of reusable software from “Measuring Software Reusability” [32] ©1994 IEEE.
As illustrated in Figure 4.1, the most vital aspect of reuse is ease of understanding. It is as important to software as it is to ontologies. Moreover, ease of understanding is seen through the types of documentation stored about each ontology and the description it provides. This information can be represented as the abstract. This is a direct parallel between software and ontology domains. Another factor that is required is the Change History as it allows the users to understand the change between versions of the software. This may effect its integration or use within a large system. This parallel is accurate within the domain of ontologies as a change in an ontology can potentially have a dramatic impact on how the ontology is used within a system and the information that it outputs. Further details of this could be seen in Chapter 3 Section 3.3.

Dependencies play a large part in software, as it is an underlying requirement for any software to run. When relating this to the ontology domain, ontologies have the same kind

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>Provides a clear, concise description of the component.</td>
</tr>
<tr>
<td>Change history</td>
<td>Describes changes to the code, who made them, the date of the changes, and why.</td>
</tr>
<tr>
<td>Dependencies</td>
<td>Describes prerequisite software and other software the component uses.</td>
</tr>
<tr>
<td>Design</td>
<td>Describes the internal design of the code and major design decisions.</td>
</tr>
<tr>
<td>Interface</td>
<td>Describes all inputs, outputs, operations, exceptions, and any other side effects visible to the reuser</td>
</tr>
<tr>
<td>Legal</td>
<td>Provides a summary of legal information and restrictions, such as license and copyright information.</td>
</tr>
<tr>
<td>Performance</td>
<td>Describes the time requirements, space requirements, and any performance considerations of the algorithm.</td>
</tr>
<tr>
<td>Restrictions</td>
<td>Lists any situations that limit the usability of the component, such as nonstandard compiler options and known side effects.</td>
</tr>
<tr>
<td>Sample</td>
<td>Provides a usage scenario showing how the component applies to a specific problem.</td>
</tr>
<tr>
<td>Test</td>
<td>Contains information about the test history, procedures, results, and test cases.</td>
</tr>
<tr>
<td>Usage</td>
<td>Provides helpful information on how to integrate the component.</td>
</tr>
</tbody>
</table>
of dependencies, where an ontology may depend on another ontology to define a part of the domain, or the ontology can take in parts of a different ontology to fully describe its domain as some concepts may be similar or already defined. This relationship can be seen in two ways: dependencies can go both upstream and downstream, as an ontology can be dependent on another ontology, as well as other ontologies can depend on it.

It is logical to pair **Performance** and **Design** together, as this information goes hand in hand with ontologies. Unlike software, where performance can be measured by algorithmic complexity and standard design patterns applied to the software, ontology performance is dependent on the number of classes and relations, as this impacts the time a reasoner takes to process the ontology and the duration it takes to execute queries against the ontology. For the purpose of this thesis, the case studies will combine these two attributes into one called **Metrics**.

**Organization** and structure are of upmost importance when it comes to discovery and reuse. Without structure it becomes extremely difficult to find anything. Similarly, **Documentation** helps explain the ontologies present and the ontology domain. Also, the **Sample** information can be combined within the documentation provided with the ontology.

When considering whether or not to reuse a piece of software or an ontology, one must ensure that it is of high **Quality**.

The metrics **Functional Completeness**, **Reliability**, and **High Cohesion and Loose Coupling**, are fundamental principles in software reuse. Functional completeness ensures that the software performs all necessary functions based on its requirements as well as allowing for future extension. Reliability ensures that the pieces of software complete their tasks consistently and without error. High Cohesion and Loose Coupling ensure that each component completes a specified task and that the components are not responsible for any task outside their function [32]. Similarly, in the domain of ontologies, each ontology should describe a specific domain and be only responsible for defining that domain (loose coupling and high cohesion), as well as the queries run consistently against the ontology (reliability). This will be combined into **Domain Adherence**.
Finally, **Portability** describes how portable a code base is when it is executed on one platform over another, and that the execution is consistent. When applying **Portability** to the ontology space, it refers to the which language each ontology is defined syntactically (DAML, OWL, etc.) as well as ensuring consistent execution across different reasoners. Furthermore, portability also looks at the semantic languages that the ontology is defined in (English, French, etc.) as the nuances of different languages may change the names of the relationships or classes within the ontology.

Ultimately, each case study will be looking at whether an ontology can capture these attributes as well as ensure reuse. It is fair to make this connection, as software and ontologies are both functional, and many of these metrics have parallels with the domain of ontologies. We have translated and aggregated the metrics into a list of helpful questions to see if the systems we view help apply this information to these test cases. We can then map these attributes into the ontology space. The attributes are mapped into a question list to act as guidelines, as addressed and used by the case studies. The checklist can be seen below:

1.) **Abstract:** Does the methodology provide a description of each ontology within the system? If so, what kind of information is captured within this description?

2.) **Change History:** Does the methodology provide a means to describe and outline change between each version of the ontology? If so, what kind of information is captured?

3.) **Dependencies:** Does the methodology have a means by which to describe the dependencies of an ontology? Does it have a means to denote which ontologies rely on another ontology?

4.) **Metrics:** Does the methodology have a means to provide metrics about each ontology? How many classes exist? How many relations?

5.) **Organization:** Does the methodology have a means to organize the ontologies in a logical manner? How granular is the classification?
6.) **Documentation:** Does the methodology have ability to access documentation and additional information about the ontology?

7.) **Quality:** Does the methodology have some sort of QA process to ensure the ontology’s correctness and completeness?

8.) **Domain Adherence:** Does the methodology ensure that the ontologies are of a singular domain, and are describing a specific part of that domain?

9.) **Portability:** Does the methodology have a means to allow for ontologies to be described in different languages as well as different syntactical languages? Does it have a means to track the differences between syntactical representation? Does the methodology describe the differences between the various representations.

### 4.2 Case Study 1: Date Time Ontology

#### 4.2.1 Synopsis

In this case study we will examine how each methodology handles the discovery and reuse of a foundational ontology. Specifically, this case study sets out to see how each methodology handles foundational ontologies; are they present, and what advantages and disadvantages each methodology has relative to foundational ontologies. We will setup the case as follows:

“A software engineer is looking for an ontology to capture date and time information to help translate and format time from various regional formats as this will be part of an enterprise to enterprise system. As this ontology will be used to aid in commerce transactions, it must be well maintained, high quality, and relatively stable. As this will be part of a near realtime system, the engineer must be mindful of the performance of the ontology. The ontology syntax representation is not important as the underlying architecture supports almost all ontology representation languages.”

With this case study, we can see that the user is strictly a user of an ontology and not necessarily a domain expert. With this being said, this implies that the user will be rely-
ing heavily on the methodologies to provide much of the information necessary to use the ontology, along with the methodology to ensure the quality of the ontology. Hence, we can establish the key criteria from the checklist, which is important; for this case, **Abstract**, **Change History**, **Dependencies**, **Metrics Documentation**, **Quality**, **Domain Adherence** are extremely important to the engineer as, this information will guide them in deterring whether the ontology will be usable and of high quality relative to the criteria set out in this case. Though in general item, **Portability** is normally important when considering the reusability of an ontology, in this case it is less important as the case mentioned that the different languages were not important, relative to the system.

Thus, we will apply the checklist to each methodology in the following section.

### 4.2.2 Analysis Using Keyword Search

This case will be examining the paradigm of a non repository repository. It will be the baseline case using existing search technologies to see if it is possible to meet all the reusability criteria. The setup for this case was simple. To set up the browser in a way to minimize the impact of the previous search history, the browser was placed into “incognito mode”. This mode disables all cookies and cache as well as logs the user out of any accounts the search engine may use to help tailor historical search to the user’s favour. Through the research presented in this thesis, the search results were highly tailored to specific ontology pages unrelated to our search and we wanted to create a vanilla environment as if the user was searching for the ontology for his first time. For our initial attempt, we used the keywords **“date ontology”** when using the search engine. The initial results are seen in Figure 4.3. As we can see there are no usable results present within the first set of keywords that we used. Furthermore, the search engine recommends the keywords **“data ontology”**. When clicking on this we see the results in Figure 4.4. As illustrated, this yielded no results relevant to our initial search criteria. Furthermore we can see the results are less relevant than our initial result. After several different keyword combinations and trials we discovered **“time ontology owl”** produced promising search results. These results
can be seen in Figure 4.5. The specific ontology which would be ideal for this situation was the second result. Now with an ontology found, we can analyze the system for reusability with the checklist defined above. The link to the page used for analysis can be found here [19] and the link to the ontology can be found here [19].

Figure 4.3: Google search results when using the keyword “date ontology”.

Applying the Checklist

We will now systematically apply the checklist to this methodology.
Abstract: Does the methodology provide a description of each ontology within the system? If so, what kind of information is captured within this description?

This system provides a minimal abstract in the search abstract. It simply provides the first few lines of the linked ontologies abstract on the page writeup. This is not guaranteed by this methodology as the search engine only discovers existing content and has no control over the content itself. For this specific ontology however there is an additional abstract present describing what the ontology does and the languages that it is described in.
Figure 4.5: Google search when using the keyword “time ontology owl”.

Change History: *Does the methodology provide a means to describe and outline change between each version of the ontology? If so, what kind of information is captured?*

This methodology does not directly provide any sort of method for change history. It is entirely dependent on the content provider. In the current case, there are no direct links to past versions of the ontology although, earlier versions are mentioned in the document.

Dependencies: *Does the methodology have a means by which to describe the dependencies of*
an ontology? Does it have a means to denote which ontologies rely on another ontology?

This methodology does not directly support any form of dependencies as it has no control over the content provider. However, in this specific case, it can be seen that the document provided describes that this ontology is independent and can be used in concert with a few specific ontologies described. Additionally, it is important to note that the document has no standardized way of describing dependencies, it was only described here under the section “General Issues”.

Metrics: Does the methodology have a means to provide metrics about each ontology? How many classes exist? How many relations?

This methodology does not directly support any form of metrics as it has no control over the content provider. However, in this specific case it can be seen that the document has descriptions of various classes present within the ontology but does not have any direct metrics to the number of classes and relations present in the class.

Organization: Does the methodology have a means to organize the ontologies in a logical manner? How granular is the classification?

This methodology does not have any means to organize the ontologies in a logical manner, as it can only apply keyword searches and order results in a manner of inferred relevance.

Documentation: Does the methodology have ability to access documentation and additional information about the ontology?

This methodology does not ensure any means of documentation, however in this instance, detailed documentation about the ontology is present. It gives a description of what the ontology does, what its appropriate usages are, and who published it. Additionally it gives example usages. Once again this level of information is not guaranteed by the keyword search methodology.

Quality: Does the methodology have some sort of QA process to ensure the ontology cor-
rectness and completeness?

This methodology has no method to ensure quality. As a user, we are required to gauge the quality of the ontology. In this case the ontology is published and maintained by the W3 organization. However this is not ensured, as another search for a different ontology may result in an ontology of poor quality.

Domain Adherence: Does it the methodology ensure the ontologies are of singular domain, and are describing a specific part of that domain?

This methodology does not ensure domain adherence as it has no control over the content presented.

Key Observations

We can see from the checklist above that keyword search does not give much to support the discovery and reusability of ontologies as defined by the metrics. Many of these metrics are not enforced, and thus for this case it was fortunate for us that much of this information required was present. If we were to search another ontology, which may not have been as common, or as broadly used it would likely be in a scenario where none of the information necessary to determine the usability of an ontology would be given. Furthermore, as discussed in Chapter 2 this methodology only does keyword search on ontology names. Specifically, in this case, it searches across pages which link to the ontology and does not necessarily have any information related to the ontology itself other then the file name. It could be possible that a file is misnamed or even given an id number and keyword the search would outright fail, if the page had no relation to the ontology other then a link to the file. Hierarchical organization of ontologies in this methodology does not exist by its definition, as it searches across its database to find information most related to the keyword.

However, it may be possible to optimize the search query to help trim down results to file type by only returning “.owl” files, or further increasing the specificity of the keywords. Nonetheless, the rest of the reusability requirements are still left unmet and unenforced.
4.2.3 Analysis Using BioPortal

In the previous section we looked at the methodology of using a raw keyword search with a standard search engine. In this section, we will be looking at the discovery and reusability of ontologies produced by a system designed to capture, store, and share ontologies. Furthermore, we will examine BioPortal and the results produced against the checklist and the case study we are looking at. For this experiment, we went to the BioPortal website located at [http://bioportal.bioontology.org/][13]. What is presented can be seen in Figure 4.6. From there, we selected the explore option followed by entering the keyword “time”. The subsequent results can be seen in Figure 4.7. From the list the top ontology seemed to be the most relevant to our search. Once selected we were presented with the information seen in Figure 4.8. At this point we were able to apply the usability heuristics checklist to the system.

![BioPortal](https://example.com/bioportal_screenshot.png)

Figure 4.6: The information available in BioPortal for Time Event Ontology.
Figure 4.7: The information available in BioPortal for Time Event Ontology.

Figure 4.8: The information available in BioPortal for Time Event Ontology.
Applying the Checklist

We will now systematically apply the checklist to BioPortal

Abstract: *Does the methodology provide a description of each ontology within the system? If so, what kind of information is captured within this description?*

BioPortal does not provide a direct method for giving an abstract of the ontology. However, it does provide generic information about each ontology. For this instance the data provided includes a generic acronym, the visibility of the ontology, an extremely brief description of the ontology, and a status of the ontology (in this case Alpha).

Change History: *Does the methodology provide a means to describe and outline change between each version of the ontology? If so, what kind of information is captured?*

BioPortal supports a list of the different versions of the ontology contributed to the repository. Additionally, two of the links to previous versions produce Internal Server Errors. Furthermore, there is no reference to what the changes entail within the ontology. After investigating further, the notes section was bare and did not mention anything about changes between versions of the ontologies.

Dependencies: *Does the methodology have a means by which to describe the dependencies of an ontology? Does it have a means to denote which ontologies rely on another ontology?*

BioPortal does not provide any notion of dependencies. BioPortal does have a notion of which ontologies depend on this specific ontology. This is done through the mapping section. It gives the ontologies which use the Time Event Ontology. It also gives a list of one to one class mappings. However, it does not give much detail outside of that. It also does not define the relationship between the classes being mapped.

Metrics: *Does the methodology have a means to provide metrics about each ontology? How many classes exist? How many relations?*
BioPortal provides metrics via the metrics panel on the summary page. It lists the number of named classes within the ontology, the number of properties in the ontology, the number of individuals within the ontology, as well as the maximum depth of any relationship.

Organization: *Does the methodology have a means to organize the ontologies in a logical manner? How granular is the classification?*

BioPortal by nature specifically specializes in the biomedical domain however it does not have an overall structure for ontologies. They are not organized in any specific manner. Access to the ontologies it is done via keyword search. Each ontology is stored independently of each other and are not grouped in any logical manner.

Documentation: *Does the methodology have ability to access documentation and additional information about the ontology?*

BioPortal does not have any documentation related to this ontology. However, there is a section in the summary that allows for a link to documentation on the ontology, although the system has no method to enforce this.

Quality: *Does the methodology have some sort of QA process to ensure the ontology correctness and completeness?*

The primary modality for quality is present through the metrics provided on the summary page to allow the user to understand the composition of the ontology. Additionally, BioPortal provides users with metrics on the number of views. No other means of quality is enforced by the system.

Domain Adherence: *Does it the methodology ensure the ontologies are of singular domain, and are describing a specific part of that domain?*

BioPortal has no methodology to ensure domain adherence as the user can potentially upload any ontology they wish.
Key Observations

When reviewing the metrics applied to BioPortal, it became clear that it fared better than the simple keyword search. First, we must address that the ontology discovered using BioPortal does not truly represent Date Time information. It only has one set of classes and subclasses related to the users search and this was quite incomplete. In this scenario, a user would continue looking for another ontology that would likely better fit the desired problem they are trying to solve. However, at the time of writing this document this ontology seemed to be the best fit for the criteria outlined by the case study.

Putting aside the ontology not defining the necessary domain information, the analysis still stands when examining the types of meta information being presented by the repository. When we take a look at access and discovery of the ontology, it still was found via keyword search. The same deficiencies described in the first example of the case study is still relevant here. While some additional tools are added, this still does not address the key issues with keyword search. Through this search facility, the only resemblance of domain adherence is when keyword categorization filtering is present. This is done through categorizations added in by the user when uploading the ontology, and thus this system does not guarantee this information is present.

When considering abstract and associated documentation, BioPortal does give facilities for the user to add this when uploading an ontology. However, much of this information is absent in this case. Specifically, when looking at the abstract for the ontology it is not present. It has a general summary for the ontology, but much of this information is missing. Furthermore, there is no documentation link present. The issue is not the actual terseness or missing link, it is the repository itself does not enforce these to be present, and without that we are missing key characteristics necessary for reusability.

This theme continues throughout all the metrics provided by the repository. Though some pertinent information is present the lack of resolution in detail exists too. Especially, when we look at the Change History metric, this information is outlined within the system summary page as in there are multiple versions of the ontologies, but there are no facilities
to contrast what changes have occurred and what classes may have been deprecated.

BioPortal does, however, have a notion of Dependencies. It shows all ontologies within the repository that reference the selected ontology. It also shows the class mappings. But it does not show the direction of the mapping, i.e. is the ontology dependent on the Time Event ontology or is the Time Event ontology dependent on the other ontology. Though this feature does help in regards to further reusability, it is not intrinsically tied to the repository methodology nor would another repository have to implement such a feature.

Another such feature implemented in BioPortal, is that of metrics. It creates empirical metrics based on the counts of classes and substructures in the ontology. This helps inform the user of the size of any given ontology as well as an overview of the complexity of the ontology. Once again this is a feature implemented by BioPortal and not a characteristic present in a repository. Which leads into the metric of quality. The QA process imposed within this repository puts the onus on the user of the ontology. The only quality assurance provided is the metrics on the summary page. This leaves the user to have to analyze on their own to determine the quality of the ontology and if it captures the necessary information adequately.
4.2.4 Analysis Using SOL

Now we will apply the same metrics to the SOL. Since this library was defined by the thesis, we will have to make the assumption that the date time ontology is present within the library. Specifically, we used the W3C time ontology found at [https://www.w3.org/TR/2006/WD-owl-time-20060927/][19]. We will then apply the metrics to the meta ontology as described in Chapter 3. Specifically, we will assume an ontology is present similar to the one found within Section 4.2.2. We will go through each member of the checklist and examine if the meta ontology has the facilities to capture specific information as well as if it is possible to retrieve the information via queries.

```
This ontology defines both date and time formats. It additionally provides
the necessary conversions between the various date and time formats.
```

Figure 4.10: An example of a potential description that could appear in the SOL.
Applying the Checklist

We will now apply the checklist to the SOL

Abstract: *Does the methodology provide a description of each ontology within the system? If so, what kind of information is captured within this description?*

The SOL provides the mechanism necessary to describe what the ontology is as well as provide additional documentation. Simply by querying the ontology we want, we can see the “hasDescription” property on the ontology. In our test example, the description can be seen in Figure 4.10. It would capture the primary use and function of said ontology. This description could be as large as necessary as it is captured via a data property that is of type string. That being said current ontology representation languages do not have the ability to define a minimum string length so it is possible that this field could be empty. This can be checked and enforced during the review by committee process as explained in Section 3.4.2.

Change History: *Does the methodology provide a means to describe and outline change between each version of the ontology? If so, what kind of information is captured?*

The change history capture by the SOL is via the versioning class structure. It allows the ontology described within the Meta Ontology to have versions as well as changes that occur in the ontology. Specifically, a version within the SOL is captured via object property. Additionally, the types of change that have occurred from the version are captured in the Change class hierarchy. The class hierarchy can be seen in Figure 4.11 and the object property can be seen in Figure 4.12.

Dependencies: *Does the methodology have a means by which to describe the dependencies of an ontology? Does it have a means to denote which ontologies rely on an ontology?*

The SOL provides dependencies relations that are described through the object property isDependent and hasDependencies on. This will link ontologies
that are dependent or depend on this ontology. In this case when looking at the
dateTimeOntology we can see that in this example the only one other ontology currently described within the SOLs meta ontology is the HTML ontology. However in the SOLs’ meta ontology the hasDependency and isDependencyFor is an inverse relation. So any ontology can see which ontology it depends on and which ontologies depend on it.

Metrics: *Does the methodology have a means to provide metrics about each ontology? How many classes exist? How many relations?*

The metrics provided within the SOL are very similar to that of the metrics included in the following data properties numberOfClasses, numberOfDataProperties, numberOfObjectProperties, and numberOfIndividuals.

Organization: *Does the methodology have a means to organize the ontologies in a logical manner? How granular is the classification?*

The ontologies are organized in a hierarchical structure defined OntologyDomain. This allows the user to search for the ontology based on domain.

Documentation: *Does the methodology have the ability to access documentation and additional information about the ontology?*

There is a relation within the meta ontology that relates to documentation. This also provides links to the associated documentation about the ontology. This is in the form of links to documentation via the Documentation class which is a subclass of the Ontology Properties class.

Quality: *Does the methodology have some sort of QA process to ensure the ontology correctness and completeness?*

Quality is ensured by entry into the ontology library. Combined with the metrics defined, each ontology is vetted before entry into the library. This ensures only ontologies of high quality are associated. Furthermore, information about the author is present within the ontology. The author additionally has a link to the organization the author belongs to as well as their role within the
organization. This transitively links an ontology to an organization. This can lead to certainty of which organization is responsible for an ontology as well as which user is responsible for its maintenance and the associated relationships within the SOL.

Domain Adherence: *Does it the methodology ensure the ontologies are of singular domain, and are describing a specific part of that domain?*

Due to highly structured nature of the ontology it is required that each ontology belong to a specific domain. Additionally the review process ensures that the ontology is of the correct domain and is categorized accordingly.

![Figure 4.11: The class hierarchy for change as described by the SOL.](image)

![Figure 4.12: An example of the change version for the date time ontology in the SOL.](image)
Key Observations

When applying the same heuristics to the SOL, we see that it meets the requirements set by many of the heuristics through the information captured by the meta-ontology. Many of the metrics are enforced through the structure of the meta-ontology itself. In this we see the primary difference between the SOL and the other methods. This is because the structure of the meta-ontology can enforce the existence and cataloging the various aspects of the ontology being housed by the library. This ensures completeness and satisfies all metrics related to ease of understanding, specifically, Abstract and Change History. Additionally, the SOL captures documentation via the Documentation class and url link to the associated documentation for that ontology. Though the SOL can ensure that a URL is present and in the correct format it cannot guarantee that it is accurate or leads to the correct content, this task will fall upon the maintainer of the ontology as well as the committee in charge for that portion of the SOL as described in Section 3.4.2.

The SOL and its meta-ontology also capture several key metrics. Specifically, the ontology captures information on number of classes, number of object properties, number of date properties, and number of entities. This allows the user to make various determinations on the size of the ontology. However, this falls short of the extent of metrics offered by BioPortal. In contrast to BioPortal the SOL ensures the metrics are present, as it will not be allowed into the library via the review process without these metrics being present.

Quality is guaranteed via the review process, as well as the knowing the organization and author responsible for the ontology being viewed. Finally domain adherence is done through two methods: one is the categorization through the meta-ontology and the other is the specific category each ontology belongs to. This allows the user to quickly understand which domain the ontology is a part of. To ensure that each ontology has only classes of a specific domain and is only responsible for that one domain, the quality review process checks these characteristics. This is performed before admittance to the SOL.
4.2.5 Findings and Discussion

Based on the analysis presented above we can see that the Standard Ontology Library outperforms the methodology of a repository. It is obvious that the SOL was designed to capture both domain specific and domain independent ontologies while the repository analyzed in this case study was designed for biomedical ontologies. However, where these two methodologies diverge most is the enforcement of the information being presented. When looking at BioPortal it is possible for any individual to contribute an ontology. Once contributed, the ontology can be accessed by anyone. Moreover, none of the additional information is required to be populated. This makes the checklist we applied to see if the repository would meet the requirements for reuse, differ from ontology to ontology found. This could potentially lead to ontologies on various pages being highly well documented while other ontologies could have next to no information. This kind of disparity between the quality of information present detracts from the overall trust in quality of any individual ontology. It would also make it difficult for a user to make a determination of quality of any given ontology in the system. Looking at the case, we know the user needs to ensure the quality of the ontology as it will be used within a system’s critical operation, and considering the inconsistency of information within the repository, the determination of reuse cannot be done simply by reviewing the records within the repository.

Conversely, the SOL does not allow any ontology to enter the library unless it has been reviewed. Furthermore, ontologies being added are not allowed into the core unless they have been thoroughly scrutinized. This ensures the quality of any ontology present within the repository. This allows any user to intrinsically trust that it has been adequately scrutinized to appear in the library. Moreover, the disparity in metrics available between BioPortal and SOL can be easily remedied by adding additional data properties to the meta-ontology. Once present in the meta-ontology, all the ontologies’ meta information would be aggregated, and when the next version of the library is released all this meta information would be present, as well as guaranteed to be present. Additionally, this information could be automated via queries on the ontology to ensure that it is present.
Finally, it would be remiss not to review the keyword search methodology. This methodology does not ensure that any of the information is present. Every search could potentially produce great results, or it could produce terrible results, it is truly unknown until the search is executed. Furthermore, the ontologies found through the search may be of high or low quality, and it is difficult to know who the publisher of the ontology is. For example, it is possible for a site to host an ontology, however the owners of the site were not the individuals who generated the ontology in the first place. This makes it difficult to track down ownership to determine if it is suitable for reuse. Consequently, in this modality many other heuristics may or may not be present. In relation to this case study, this would be the worst possible choice to locate and an identify an ontology for reuse.

4.3 Case Study 2: Domain Specific Ontology Discovery

4.3.1 Synopsis

In the previous case study, the ontology we set out to discover was more geared to a generic repository, and was maybe not geared well to BioPortal which was highly domain specific. Hence, for this case study we will attempt to discover an ontology in the biomedical field. This case will be tailored to best suit the ontology repository and we will then compare and contrast against the defined metrics once again. The same assumptions for the SOL will be present as in the last case study. The case is as follows:

“An immunologist is looking for an ontology to capture gene information to help within their new viral vector simulators. These simulators require a variety of ontological inputs such as, owl, loom and rdf as each component was developed by different research groups and intrinsically depend on different ontological needs. Computer resources and time constraints are not a big issue when using this ontology in concert with these models. The ontology must be of high quality and from a trusted source as the results from running these models will be used to underpin various studies that are currently being researched.”

With this case study we can see that the user is both a user of an ontology as well as
a domain expert in which the ontology is used. With this in mind, the user will be able to validate the ontology, although ideally the user should be able to use the ontology based on the criteria. Additionally, validating a large ontology may take a long time delaying the research they are trying to accomplish. That being said, ideally the user is looking for an ontology that is of high quality and from a trusted source. Furthermore, the user is not interested in the ontology’s performance. Hence, for this case study we will be focusing on the following metrics: Abstract, Dependencies, Organization, Documentation, Quality, Domain Adherence and Portability. These metrics are required to determine the quality of the ontology, as well as the different language representations of them. Absent from this list are Metrics and Change History, as neither performance or long term maintenance are not necessarily important in this case.

As seen in the first case study, the keyword search methodology would have the exact same results regardless of domain. It will not be necessary to apply the checklist to keyword search as the results will not change.

Thus we will apply the methodology to BioPortal and the SOL in the following sections.

![Gene Ontology](image)

Figure 4.13: The information available in BioPortal for the Gene Ontology.

66
Figure 4.14: The information available in BioPortal for Gene Ontology related to versioning and views.

Figure 4.15: The information available in BioPortal for Gene Ontology related to projects using this ontology.

4.3.2 Analysis Using BioPortal

Once again we will be looking at the discovery and reusability of ontologies produced by a system designed to capture, store, and share ontologies. In this section we will examine
Figure 4.16: The information available in BioPortal mapping the Gene Ontology to other ontologies.

Figure 4.17: Example of one such mapping between classes in the Gene Ontology and the Adverse Event Reporting Ontology.

BioPortal again and the results produced against the checklist and the case study we are looking at. For this experiment we went to the BioPortal website located at http://
Once again, when entering the Gene Ontology into the keyword search we were presented with the Gene Ontology immediately. We selected the ontology, and you can see the overview presented in figure 4.13. At this point it is possible to apply the usability heuristics checklist.

Applying the Checklist

Abstract: Does the methodology provide a description of each ontology within the system? If so, what kind of information is captured within this description?

In this case BioPortal does not give a clear abstract for the ontology. We can see in Figure 4.13 that the description field is blank.

Dependencies: Does the methodology have a means by which to describe the dependencies of an ontology? Does it have a means to denote which ontologies rely on an ontology?

As in the previous case study, BioPortal does not provide any notion of dependencies in this case. BioPortal does have a notion of which ontologies depend on this specific ontology. This is done through the mapping section as can be seen in Figure 4.16 and Figure 4.17. It gives the ontology, which uses the Time Event Ontology, as well as others. It also gives a list of one to one class mappings. However, it does not give much detail outside of that. It also does not define the relationship between the classes being mapped.

Organization: Does the methodology have a means to organize the ontologies in a logical manner? How granular is the classification?

In this case, BioPortal provides a category for the ontology: “Genomic and Proteomic”. But there is no mechanism to explore other ontologies of the same type through the view presented.

Documentation: Does the methodology have ability to access documentation and additional information about the ontology?

In this case, BioPortal provides a link to an external url with detailed documentation about the ontology. The documentation is very detailed and informa-
Quality: Does the methodology have some sort of QA process to ensure the ontology correctness and completeness?

For this ontology has a new section present called Review. This can be seen in Figure 4.13. The review shows that the Gene Ontology, has been given 4/5 starts on the criteria Usability, Coverage, Quality, Formality, Correctness, and Documentation. Furthermore, it details that it applied 102 criteria to the ontology. It was unclear what those criteria are and how they translated to the star ratings on the side. In addition they provide a link to where the review was done. However, this link was broken at the time of writing this document. Finally, there is no indication which version of the ontology the review was applied to.

Domain Adherence: Does the methodology ensure that the ontologies are of singular domain, and are describing a specific part of that domain?

There is no guarantee of domain adherence.

Portability: Does the methodology have a means to allow for ontologies to be described in different languages as well as different syntactical languages? Does it have a means to track the differences between syntactical representation? Does the methodology describe the differences between the various representation.

BioPortal reports conflicting information on the portability of the ontology. Specifically, we can see in Figure 4.13 that the format is of “OBO” however this is different from what is reported in the “submissions section” which has formats of “OBO”, “CSV”, “RDF/XML”. This can be seen in Figure 4.14. It is unclear which is correct. It does not have any means to track the differences between any of the syntactical representation.

Key Observations

Unlike the previous case study, the amount of information present for this ontology is much more complete then that of the first case study. The versions of the ontology are
well displayed as seen in Figure 4.14, as well as a through lists of projects that employ
this ontology as seen in Figure 4.15. It also provides a review of the ontology. From the
surface this looks like it has everything needed to make a decision on whether or not to
use this ontology. However, this is not the case when applying the checklist heuristics once
again. Furthermore, many of the same conclusions derived from the first case study can be
determined here.

The most obvious is that there is no methodology for domain adherence. The ontology
may be a related directly to the biomedical field, however this was not ensured by the
system. The user was able to upload this ontology just as any user was able to ontology
on any particular topic. Furthermore, a defined abstract for the ontology is missing. Only
through the documentation link do you fully understand how the ontology is composed and
what its purpose is. An additional source of ambiguity is the Portability heuristic, as the the
page defines that the ontology is represented in a singular language (OBO) then conflicting
formats in the version section. This inconsistency makes it unclear which formats are truly
supported. Additionally, there is no mechanism to signify differences between the format
representations.

Finally, when considering quality, once again the system does not provide any direct
enforcement and requires the user to judge quality on their own. The review is helpful when
present, however, it is difficult to understand what criteria each user applied to determine
the quality of the ontology. The review explained that the ontology was thoroughly checked
and scrutinized under a well conceived checklist however, the links no longer worked, nor
was there an explanation of how they translated the review to the star rating system present
in BioPortal. This all being put aside, the most troubling portion of the review mechanism
present is that there is no way to link to the version of ontology that has been reviewed.
It is quite possible the ontology is no longer being maintained or the quality has decreased
since the last publishing of said review.

This all culminates to enforcement, which is a recurring theme in both case studies.
Repositories simply have no methods to enforce any of these criteria as the contribution
model is that of people contributing and determination of fitness is done upon access and use. Contributors may take great care in documenting the ontology as well as ensuring it is up to date, as seen in this case. The consortium of individuals took great care in publishing a quality ontology. However, if we take another example out of the repository as seen in case one next to no care was given to the ontology and the user had to make all determinations for themselves. Additionally, since much of the information is contributed from external sources without being peer reviewed and without a well documented process of this review, it is difficult to know if the information present is of quality.

4.3.3 Analysis Using SOL

For this case study we inserted the Gene Ontology into the SOL meta-ontology. We then applied the same checklist as in the previous section. We made the assumption the ontology was already reviewed for quality as well as categorized appropriately. These tasks would of been done by the review committee before the ontology was even placed within the SOL. It was not possible to reconstruct the differences between prior versions, however the capability is present within the SOL to track that information via the Change class hierarchy. We will once again apply the same checklist to the meta-ontology.

Applying the Checklist

Abstract: *Does the methodology provide a description of each ontology within the system? If so, what kind of information is captured within this description?* As in the first case study the ontology entity for the Gene Ontology has the data property description which describes the overall use and purpose of the Gene Ontology. It makes it clear what the ontology is for. Additionally the ontology class and all its relations give a overall view to all the components of the ontology any necessary information the user would inevitably want to see before use. You can see many of the relations in Figure 4.18.
Dependencies: Does the methodology have a means by which to describe the dependencies of an ontology? Does it have a means to denote which ontologies rely on an ontology?

The SOL provides dependencies relations that are described through the object property isDependent and hasDependencies. This will link ontologies that are dependent or depend on this ontology. In this case when looking at the Gene Ontology there are no other ontologies that the gene ontology depends on or is dependent on.

Metrics: Does the methodology have a means to provide metrics about each ontology? How many classes exist? How many relations?

As in the previous case study the SOL provides statistics on the Gene Ontology via the Ontology Properties subclass Statistics and the data property relating it to the ontology hasStatistics. The same statistics are available as in the first case study. We can see this in Figure 4.20.

Organization: Does the methodology have a means to organize the ontologies in a logical manner? How granular is the classification?

The SOL provides the classification of the ontology by defining the domain under Genetics within the Industry Specific on Biology structure. The ontologies are all divided by domain. This is the same as seen in the first case study as well as Chapter 3.

Documentation: Does the methodology have ability to access documentation and additional information about the ontology?

Much like the previous case study, SOL meta-ontology provides information and link to documentation via the Documentaiton subclass. You can see this in Figure 4.21.

Quality: Does the methodology have some sort of QA process to ensure the ontology correctness and completeness?

Once again we see that quality is ensured by entry into the ontology library through the review process outlined in Chapter 3. Combined with the metrics
defined, each ontology is vetted before entry into the library. This ensures only ontologies of high quality are associated. Furthermore, information about the author is present within the ontology. The author additionally has a link to the organization that they belong to, and their role within the organization. This transitively links an ontology to an organization. This can lead to certainty of which organization is responsible for an ontology as well as which user is responsible for its maintenance.

Domain Adherence: *Does it the methodology ensure that the ontologies are of singular domain, and are describing a specific part of that domain?* Due to highly structured nature of the ontology it is required that each ontology belong to a specific domain. Additionally, the review process ensures that the ontology is of the correct domain and is categorized accordingly.

Portability: *Does the methodology have a means to allow for ontologies to be described different languages as well as different syntactical languages? Does it have a means to track the differences between syntactical representation? Does the methodology describe the differences between the various representation.*

The SOL provides basic categorization of which syntactical formats the ontology is represented in. It does not however have any criteria to denote which syntactical langue is the primary one, nor does it have a mechanism to catalog discrepancies between representation.

Key Observations

Much like the previous case study, the SOL was able to meet many of the criteria required to promote reusability. Specifically, we see the structure of the meta ontology and the information captured by it well suited to almost all criteria accessed within the checklist. We can see that it has a descriptive abstract, as well as authorial information. The ontology
Figure 4.18: The relation between the Gene Ontology and various descriptors and metrics. Also has a notion intellectual ownership through the IP class, which may not be directly covered by the checklist but it provides additional information for determination of use (this can be seen in Figure 4.19). This is important to this case study as the results using this ontology may want to be published publicly or redistributed, this knowledge may effect the decision of whether or not to use the ontology.

Many of the additional criteria present within the checklist were covered and enforced by the ontology. This shows that the ontology library is suitable to both domain independent and domain specific ontologies. Regardless of domain the criteria and results of the analysis are the same and consistent. Additionally the enforcement and validation of data completeness remains true as if any of the data is not present or incorrectly categorized the ontology would become inconsistent.

Through this analysis, there is one criteria the SOL does not entirely meet: Portability. The meta-ontology employed by the SOL does have a notion of syntax language representations through the SyntaxLanguage class and the object property hasSyntaxLanguage although the library identifies all languages the ontology is represented in. However, there is no notion in the meta-ontology to capture the differences in representation and issues that may arise from selecting one implementation over another.
Figure 4.19: Properties of the Gene Ontology present within the meta ontology.
Figure 4.20: The relation between the Gene Ontology and its statistics.

Figure 4.21: Properties of the Gene Ontology documentation.
4.3.4 Findings and Discussion

After completing this case study it becomes clear that once again the SOL is the preferred modality to promote reuse of ontology. Although, BioPortal captured more information about the ontology for the user to determine if it is of quality and use. The primary issue is validity of the information being presented as well as consistency of information between each ontology. After a review of both these modalities, you can see the disparity between the philosophy of each modality. It is clear BioPortal puts the onus on the users of the ontology to ensure quality as well as share quality with the community, however, this leads to several problems that are emphasized by the reusability metrics used in the checklist. By shifting the onus to the user, there is is no requirement for the system to enforce that the necessary criteria to formulate a decision are present. The primary focus of the system becomes storage and retrieval. This fundamental difference lead to the conclusion derived from both cases studies, as well as explains the inconsistency between the information between each ontology stored in BioPortal, and the consistency seen between ontologies stored within the SOL.

Now when looking at this case study alone, since the ontology that was required to be discovered was from the biomedical domain BioPortal should of performed exceptionally. However, many checklist criteria were absent or incomplete. It is true the information presented for the Gene Ontology in the second case study was far more complete then that in the first case study; it still lacked many of the criteria necessary for reuse, and this speaks to the inconsistency found in repositories. Repositories are designed to share items amongst a group of individual, and it is very good at storing and distributing syntactic objects, however the methodology falls short when it has to convey semantics. By looking at ontologies as functional structures we were able to take inspiration from software libraries as described in Chapter 2 and Chapter 3, and it was possible to derive a solution that can both meet the necessary syntactic storage needs but as well convey the necessary semantic information to foster reuse as seen in these case studies.
Chapter 5

Conclusion

This thesis sought to determine whether the metaphor of a library was better suited to facilitate ontology reuse over that of a repository. We initially reviewed many of the existing systems and determined the major gaps. It became clear that many of these systems expected the user to validate the ontology that was housed in each system, while the system simply warehoused each ontology as is. Repositories often treated ontologies as files or taxonomies strictly focusing on the syntactic information present, but this alone is not what defines an ontology. Once we take into consideration the semantic aspects of an ontology we can no longer view it simply as structured relational taxonomies but as functional structures with purpose. This lead us to the realization that the metaphor of a library would be much more suitable to define and house related functional structures.

Based on this research we developed the framework and general structure of an ontology library. We defined the necessary information which would be required to be captured by such a library as well as means to ensure quality. It was clear that this information would need to be structured, and the the best way to capture information of an ontology and enforce its presence was through an ontology. This lead to develop the SOL meta-ontology the backbone necessary to exist for an ontology library.

To validate the design of the standard ontology library and its suitability for reuse,
we carefully selected case studies, where we compared and contrasted the ontology library against its repository contemporaries. For these case studies we developed a set heuristics that were applied to each system to determine if it captured the necessary information to help facilitate reuse. Through the case studies it was seen that the SOL was able to meet almost all the criteria presented, as well as enforce there existence through the meta-ontology while its repository contemporaries barley met most criterium. Fundamentally, the SOL is able to enforce the presence of much of this information through the meta-ontology as well as the review process before any ontology is admitted to the library. This enforcement is pivotal to why this methodology is more suitable to promote reuse. With the enforcement of having the necessary information present, as well as having quality review of the ontologies before admittance into the system, the onus of quality is placed on the ontology creator and not on the user of the ontology. This allows the individual to simply use the ontology after discovery as opposed to having to discover, validate, and then use. Through the case studies we also have determined that the SOL can be applied to a variety of domains with the same results. The SOL is a domain agnostic solution. Thus, we have come to the conclusion the methodology of an ontology library is more suited to facilitate reuse then that of a ontology repository.
Chapter 6

Future Work

6.1 Discovery

The SOL is capable of adequately categorizing like ontologies into a logical structures. This organization leads to a hierarchy of domains as described in Chapter 3. However, the library does not provide the means to facilitate discovery across all ontologies within the library. Through both the literature review (Chapter 2) and Case Studies (Chapter 4), it is clear that keyword search is not a suitable method for discovery. Specifically, a single keyword can map to many different ontologies or even classes within ontologies (Figure 4.9 demonstrates this relation). This leads to an incorrect assumption that the presence of class implies the ontology as whole is capturing that meaning.

Research is necessary to find the best method to help facilitate rapid discovery within the SOL. It is necessary to examine the meta properties capturing the semantic nature of an ontology and determine an efficient manner to lead a user to an ontology that satisfies its needs. ca

While the SOL is small and only captures a few ontologies, discovery is less of an issue as navigating all ontologies which belong to a specific domain is not unmanageable. However, to have this system scale to several hundred ontologies per domain it will become
next to impossible to manually discover ontologies through individual review. This is when automated discovery becomes important.

6.2 Change Management

In Chapter 3, we propose a methodology to track change within the standard ontology library. This methodology defines the severity of changes, the order in which to version the library, as well as defining structures within the meta-ontology to capture these changes. Furthermore, since these changes are stored within the same ontology for all ontologies present within the SOL, the potential that namespace collisions would occur (individuals with same name and type) is likely. Investigation into how to represent these changes within the ontology as well as mitigate the potential for namespace violations as the library grows in membership is required.

6.3 Meta Ontology Improvements

In Chapter 3, we designed the meta-ontology to capture many of the informational gaps present within the various repository systems examined in Chapter 2. However, some of the information being captured was already done by existing ontologies in the field such as the Dublin Core Ontology [1]. It would beneficial for the meta ontology to leverage these existing ontologies to capture information such abstract, author, and intellectual property rights, as it would improve the fidelity of the knowledge representation within the meta-ontology.

6.4 Validation and Review

We have defined the SOL to require a review process as well as defined the adoption model. The criterium required for the review of SOL member ontologies has not been defined. A
consistent methodology must be developed that is both domain agnostic and easily applied to all ontologies. Furthermore, the criterium for review must be concise and specific as to mitigate ambiguity to ensure as much consistency as possible with each review. Without this it will be difficult to guarantee quality of all ontologies within the library.

The author understands that due to the human aspect of the committal review there will inherently be a certain amount of inherent subjectivity; through this review process we ultimately hope to mitigate this.

6.5 Tool Sets

For ease of use and interaction with the SOL a toolset is required to help users access members and information within the library. Ideally direct integration with Protege would be the next logical step as to allow for quick access to existing members within the meta-ontology. With integration with a popular tool such as Protege it would likely foster greater adoption of the SOL. A potential system architecture for this was described in Chapter 3.

Additionally, automation tool sets should be derived to help populate the SOL meta-ontology with ontological information such a usage stats, metrics, and changes between versions of ontologies.

6.6 Language Support

In Chapter 4 it was identified that the SOL required an extension to its structure to allow for a more detailed set of information when looking at ontologies of various syntactical representation to satisfy the Portability reusability requirement. Specifically, capturing which ontological syntax is primary for the ontology, as well as what are the inherent differences between each representation.

Furthermore, a methodology is needed to be derived to capture lingual syntactical differences of ontologies present in the SOL. Currently the SOL is able to capture the language
that an ontology is defined in (English, French, etc.). It is possible that an ontology could be captured in multiple different languages, and it would be important to capture what those languages are as well as what differences in relations and semantic meaning may exist in the translated versions.

6.7 Population of the Ontology Library

Currently, the SOL is populated with a few test ontologies, as well as the associated information to satisfy the necessary requirements to perform the analysis as described in the case studies (Chapter 4). For the SOL to be released and used by others it would be necessary to populate it with a core set of common ontologies that already have been peer reviewed. Without a substantial set of ontologies the use of the SOL would be limited.
Appendix A

Meta Ontology

A.0.1 Overview of the SOL Meta Ontology

In this section we will show the all class structures present in the SOL meta-ontology.

Figure A.1: Overview of the top level class structure of the meta-ontology.
Figure A.2: Overview of the Change class structure of the meta-ontology.

Figure A.3: Overview of the Ontology Domain class structure of the meta-ontology.

Figure A.4: Overview of the Ontology Properties class structure of the meta-ontology.
A.0.2 Overview of Object Properties in the SOL Meta Ontology

In this section we will show all object properties present within the SOL meta-ontology.

![Object property hierarchy](image)

Figure A.5: Overview of the object properties present within the SOL meta-ontology.

A.0.3 Overview of Data Properties in the SOL Meta Ontology

In this section we will show an overview of the data properties present within the SOL meta-ontology.
Figure A.6: Overview of the data properties present within the SOL meta-ontology.
Bibliography


