T.R.
EGE UNIVERSITY
Graduate School of Applied and Natural Science

DEVELOPMENT OF A METHODOLOGY AND A TOOL SUPPORTING ONTOLOGY REUSE IN SEMANTIC CLOUD ARCHITECTURE

PhD Thesis

Cemil ABİŞ

Computer Engineering

İzmir
2020

T.R.
EGE UNIVERSITY
Graduate School of Applied and Natural Science

DEVELOPMENT OF A METHODOLOGY AND A TOOL SUPPORTING ONTOLOGY REUSE IN SEMANTIC CLOUD ARCHITECTURE

Cemil ABİŞ

Supervisor: Assoc. Prof. Dr. Murat Osman ÜNALIR

Computer Engineering
Computer Engineering Third Cycle Programme

İzmir
2020

Cemil ABİŞ tarafından **DOKTORA TEZİ** olarak sunulan "**DEVELOPMENT OF A METHODOLOGY AND A TOOL SUPPORTING ONTOLOGY REUSE IN SEMANTIC CLOUD ARCHITECTURE** " başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve **25.06.2020** tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:	**İmza**
Jüri Başkanı: Doç. Dr. Murat Osman ÜNALIR	. . . .
Raportör Üye: Doç. Dr. Murat KOMESLİ	. . . .
Üye: Prof. Dr. Alp KUT	. . . .
Üye: Doç. Dr. Geylani KARDAŞ	. . . .
Üye: Dr. Öğr. Üyesi Özgür GÜMÜŞ	. . . .

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ
ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca **Doktora Tezi** olarak sunduğum "**DEVELOPMENT OF A METHODOLOGY AND A TOOL SUPPORTING ONTOLOGY REUSE IN SEMANTIC CLOUD ARCHITECTURE** " başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

25.06.2020

Cemil ABİŞ

ÖZET

ONTOLOJİ YENİDEN KULLANIMINI DESTEKLEYEN YÖNTEM VE ARACIN ANLAMSAL BULUT MİMARİSİNDE GELİŞTİRİLMESİ

ABİŞ, Cemil

Doktora Tezi, Bilgisayar Mühendisliği Anabilim Dalı
Tez Danışmanı: Doç. Dr. Murat Osman ÜNALIR
Haziran 2020, 263 sayfa

Ontolojilerin yaygınlaşabilmesinin önündeki en büyük engellerden biri mevcut ontolojilerin etkin ve kolay bir şekilde yeniden kullanımının sağlanamamasıdır. Tıp, biyoinformatik ve coğrafya gibi birçok alanda geliştirilen ve yaygın kabul gören ontolojiler olmasına rağmen, ontoloji yeniden kullanım süreçlerinin ve ontoloji yeniden kullanım araçlarının yetersizliği, bu ontolojilerin yeni geliştirilen ontolojilere dahil edilebilmesini engellemektedir.

Mevcut ontolojilerin yeniden kullanılmasını sadece maliyetlerin azaltılması ve ontolojilerin daha hızlı geliştirilmesi bakımından değerlendirmek doğru değildir. Aynı zamanda bir alan içerisindeki tutarsızlıkları önleyebilmek ve birlikte çalışabilirliği arttırabilmek için bu alanda kabul görmüş veri ve bilgi kaynaklarının yeniden kullanımı gereklidir. Bununla birlikte, mevcut ve standart ontolojileri dikkate almış bir ontolojinin kalitesi ve ilgili alan içerisindeki kabul edilebilirliği de artacaktır.

Ontoloji yeniden kullanımı literatürde yer alan ve üzerinde çok sayıda çalışmanın yapıldığı önemli bir başlıktır. Tüm ontoloji geliştirme metodolojileri yeniden kullanıma değinmekte ve geliştirme sürecinde önemli bir adım olarak yer vermektedir. Büyük boyutlu ontolojileri modülerleştirmek ve yeniden kullanılabilirliğini arttırmak için çok sayıda çalışma yapılmaktadır. Ayrıca mevcut ontolojileri sorgulamak ve bir veya daha fazla kavram halinde geliştirilen ontolojiye dahil edebilmek için araç ve eklentiler de bulunmaktadır. Ancak ontoloji yeniden kullanımını bir bütün olarak yöneten bir yeniden kullanım süreci ve aracı bulunmamaktadır. Son dönemde ontoloji yeniden kullanımının mevcut durumunu değerlendirmek amacıyla yapılan çalışmalar da ontoloji yeniden kullanımı noktasındaki farkındalığın yüksek olmasına rağmen yarı-otomatik süreç ve araçların eksikliği dolayısıyla istenilen noktaya henüz gelinemediğini vurgulamaktadır.

Bu tez çalışmasının amacı mevcut ontolojilerin işlenebilmesi ve ihtiyaçlar dahilinde yeni geliştirilen ontolojiye dahil edilip yeniden kullanılabilmesi için bulut tabanlı bir yöntem ve araç geliştirerek ontoloji yeniden kullanımındaki bahsedilen bu eksikliklerin giderilmesine yönelik katkı sağlamaktır. Böylelikle ontoloji yeniden kullanımının kolaylaştırılması ve yaygınlaştırılması hedeflenmektedir. Çalışmada sadece bir yöntem önerisi değil bu yöntemin gerçekleştirimi için gerekli olan veri ve bulut mimarisi de ortaya konarak gerçek kullanıma yönelik geliştirmeler yapılmıştır. Ontoloji arama, ontoloji modülerleştirme, anlamsal ilintililik, çizge algoritmaları, NoSQL veritabanları ve Protege eklentileri gibi birçok konuda araştırmalar ve çalışmalar yapılmış ve soruna yönelik kapsamlı bir çözüm ortaya konmuştur. Bununla birlikte çevik ve yeniden kullanım tabanlı bir ontoloji geliştirme metodolojisi de önerilmiştir.

Çalışmada önerilen yöntem ve sürecin kullanılabilir bir çıktısı da elde edilmiştir. Mevcut ontolojilerin yönetimini ve modüler bir şekilde yeniden kullanımını sağlayan bulut tabanlı sunucu bileşenleri ve Protege eklentisi olarak geliştirilmiş son kullanıcı arayüzleri tez çalışmasından elde edilen ürünler olarak sunulmuştur.

Anahtar Sözcükler: Ontoloji, Ontoloji Yeniden Kullanımı, Ontoloji Mühendisliği, Ontoloji Geliştirme Metodolojileri, Anlamsal İlintililik, Ontoloji Modülerleştirme.

ABSTRACT

DEVELOPMENT OF A METHODOLOGY AND A TOOL SUPPORTING ONTOLOGY REUSE IN SEMANTIC CLOUD ARCHITECTURE

ABİŞ, Cemil

PhD. in Computer Engineering
Supervisor: Assoc. Prof. Dr. Murat Osman ÜNALIR
June 2020, 263 pages

One of the biggest obstacles to the proliferation of ontologies is the inability to effectively and easily reuse existing ontologies. Although there are widely accepted ontologies developed in many domains such as medicine, bio-informatics and geography, the insufficiency of ontology reuse processes and ontology reuse tools prevents these ontologies from being included in the new ontologies.

It is not correct to evaluate the reuse of existing ontologies only in terms of reducing costs and development time. Also, in order to prevent inconsistencies within a specific domain and increase interoperability, reuse of existing ontologies is critical. At the same time, the quality and acceptability of a new ontology that takes into account existing and standard ontologies will also increase.

Ontology reuse is an important topic in the literature, and many studies have been carried out about it. All ontology development methodologies refer to ontology reuse and include it as an important step in the ontology development process. Numerous studies have been performed to modularize large-scale ontologies and increase their reusability. There are also tools and plugins to query existing ontologies and to import them into newly-developed ontology as individual concepts. However, there is no integrated ontology reuse process and tool to handle and manage ontology reuse. Recent studies that evaluate the current state of ontology reuse also emphasize that despite the high awareness of ontology reuse, the desired point has not yet been reached due to the lack of semi-automatic processes and tools.

The purpose of this thesis study is to contribute to the elimination of these shortcomings in ontology reuse by developing a cloud-based method and tool so that existing ontologies can be processed and included into the newly-developed ontology as needed. Thus, ontology reuse processes can be facilitated.

In the study, not only a method has been proposed, but also the data and cloud architecture, which are necessary for the implementation of this method, have been developed for real ontology development purposes. Research and development have been conducted on many topics such as ontology search, ontology modularization, semantic relatedness, graph algorithms, NoSQL databases and Protege plugins, and a comprehensive solution for the problem has been presented. At the same time, an agile and reuse based ontology development methodology has been proposed.

A usable output of the proposed method and process has also been obtained. Cloud-based server components that provide management and modular reuse of existing ontologies and user interfaces developed as Protege plugins are presented as products of the thesis study.

Keywords: Ontology, Ontology Reuse, Ontology Engineering, Ontology Development Methodologies, Semantic Relatedness, Ontology Modularization.

PREFACE

In this study, an ontology development methodology, ontology reuse process, and ontology modularization method are defined to support and facilitate ontology reuse. Besides, a web application and Protege plug-in have been developed to enable the ontology reuse process to be used by end-users. It is an important contribution that the thesis study handles ontology reuse as a whole and provides a comprehensive solution starting from the ontology development methodology.

While starting the thesis, firstly, ontology reuse requirements and shortcomings have been analyzed. These shortcomings are identified by the evaluation of existing ontology development methodologies, ontology reuse processes, ontology modularization methods, and ontology reuse tools in the literature. In this way, the requirements have been clearly presented.

The first output of the thesis work is an agile and reuse based ontology development methodology. In this methodology, a comprehensive and iterative ontology reuse process is defined. Then, ontology modularization method, the most important requirement of the reuse process, has been defined and implemented. In the implementation of the modularization algorithm, methods related to NoSQL data model, graph algorithms (search algorithms and Steiner Tree problem), semantic relatedness, and semantic path restrictions have been used. Attention has also been paid to cloud data model requirements so that the architecture can provide performance and scalability.

İZMİR
25.06.2020

Cemil ABİŞ

TABLE OF CONTENTS

Page

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Page

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

LIST OF SYMBOLS/ABBREVIATIONS

Abbreviations

API	Application Programming Interface
BFO	Basic Formal Ontology
BFS	Breadth-First Search
DAML	DARPA Agent Markup Language
DFS	Depth-First Search
FMA	Foundational Model of Anatomy
GFO	General Formal Ontology
GUI	Graphical User Interface
KIF	Knowledge Interchange Format
LOV	Linked Open Data Vocabulary
MIREOT	Minimum Information to Reference an External Ontology Term
OBO	Open Biological and Biomedical Ontology
OIL	Ontology Inference Language
OLS	Ontology Lookup Service
OOR	Open Ontology Repository
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema

1. INTRODUCTION

Reuse of available resources and components in many engineering disciplines is one of the key elements that reduces costs and increases interoperability. For example; considering the software engineering discipline, using 3rd party open source libraries that fulfill a specific task significantly reduce development costs of software products. The same is also valid for ontology engineering. Reuse of existing ontologies developed in a domain will not only enhance the quality of new ontologies to be developed, but also improve the interoperability and prevent inconsistencies (Simperl, 2009).

Ontology reuse is still an open issue for ontology engineering and development. Although there are widely accepted reference ontologies in many domains, the results of the several analysis reveal that the desired level for ontology reuse is not achieved yet. The number of this kind of analysis studies have increased remarkably in recent years to emphasize the current status of reuse. One of the most important results of these studies has been revealed by Kamdar et al. (2016): Semi-automated methods and tools can make significant contributions to ontology reuse.

In order to find a solution to the ontology reuse problem, the problem must be divided into pieces and a solution must be found for each sub-problem. One of the most important parts of this puzzle is ontology development methodologies. Although many methodologies highlight ontology reuse as an important step, they do not offer methods or guidelines about how to solve this problem.

Many ontology languages, such as OWL (Web Ontology Language), used in the formal representation of ontologies, provide components for reuse of existing ontologies. For example; *owl:import* is an OWL construct that allows the existing ontologies to be completely imported to the new ontologies. While these constructs are highly beneficial for small and upper-level domain ontologies, importing large-scale ontologies, consisting of hundreds of thousands of concepts and millions of relationships, is not suitable without importing terms individually or as an ontology module.

While there are many ontology modularization methods in the literature that allow decomposition of ontologies into smaller sub-ontologies, many of these methods are not suitable for use in an iterative development and reuse process. Also, many of these methods have not been developed on a scalable cloud architecture and do not offer end-user tools. Therefore, their degree of reusability is not so high.

The problems mentioned above are just a few of the problems that outline the current state of ontology reuse. The aim of this thesis is to detail the problems in ontology reuse, to propose an ontology development methodology and to develop an ontology reuse tool for the solution of the problem. It is not possible to solve all problems of ontology reuse in a single study. Therefore, the output of this work should also be adaptable and extensible for future development.

During this study, a literature research has been conducted on many sub-fields and an ontology development methodology and an ontology reuse tool that could improve ontology reuse have been proposed. Some of these parts that form the scope of the thesis study are listed below and are visually represented as a tag cloud on Figure 1.1.

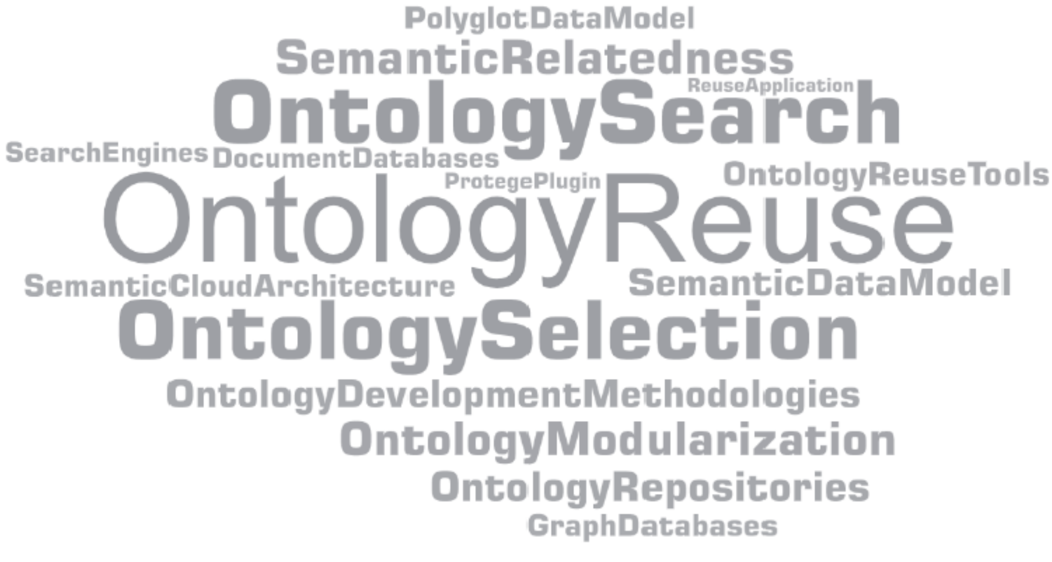


Figure 1.1: Ontology reuse tag cloud.

As can be seen from ontology reuse tag cloud, here are the important ones as listed below:

- Ontology development methodologies
  - (Fernández-López et al., 1997)

  - (Noy and McGuinness, 2001)
  - (Suárez-Figueroa et al., 2012)

- Ontology reuse tools
  - (Hanna et al., 2012)
  - (Xiang et al., 2010)
  - (Nair et al., 2011)
- Ontology repositories
  - (d'Aquin and Noy, 2012)
  - (Ding et al., 2004)
  - (Sabou et al., 2007)
- Ontology path finding
  - (Hirst and St-Onge, 1998)
- Semantic relatedness
  - (Giray and Ünalır, 2013)
  - (Hulpuş et al., 2015)
- Ontology modularization
  - (Noy and Musen, 2004)
  - (Bhatt et al., 2006)
  - (Doran et al., 2008)
  - (Ranwez et al., 2012)
- Semantic cloud architectures
- Semantic data model
- Polyglot data model
  - (Banker et al., 2016)
  - (Vukotic et al., 2014)
  - (Gheorghe et al., 2015)

1.1 Contributions

The contributions of this thesis study, which aims to develop a method and tool for ontology reuse, are listed below:

- The literature on ontology development methodologies, ontology modularization methods, ontology repositories and ontology reuse tools have created an important document that reveals the current state of the problem, even though it is not directly the output of the thesis study.
- The proposed ontology development methodology follows an agile and reuse-based approach. Ontology reuse is positioned as two sub-iterations with all reuse needs. When the existing methodologies in the literature are examined, it is thought that this approach can make an important contribution.
- The proposed and developed ontology modularization method uses path finding based on semantic relatedness. It is aimed that the extracted ontology modules are of minimum size and can be expanded parametrically according to user needs. In addition, the ontology modularization process can be easily integrated into an iterative ontology reuse process.
- Ontology reuse process and its steps have been implemented using a scalable manner on semantic cloud architecture. Thus, a real and usable product has been created for the use of ontology engineers.
- Steps of ontology reuse process have been exported with REST endpoints to enable to use by external applications. Also, a web application and Protege plugin have been developed for end users.

1.2 Contents of Thesis Report

Within the scope of this thesis, many fields have been studied to propose a solution to ontology reuse. All the contents and details studied are included in this thesis report. The organization of the thesis report is given as follows:

- In Section 2, research studies on the literature will be included and the problem will be detailed. Current studies on ontologies, ontology reuse, ontology development methodologies, ontology modularization methods and ontology reuse tools are discussed.

- In Section 3, Agile Ontology Development 101, an agile and reuse-based ontology development methodology, is presented.
- In Section 4, an ontology modularization method proposed for reusing large-scale ontologies within the scope of Agile Ontology Development 101 is explained.
- Section 5 explains how the ontology reuse process and ontology modularization method are implemented on the semantic cloud architecture.
- In Section 6, design and implementation details of ontology reuse tool, a Protege plugin that supports the steps of ontology reuse process, and web application are given.
- Section 7 elaborates the case study and the evaluation results.
- In the last section, Section 8, the study is evaluated and the future work is discussed.

Throughout the thesis report, the terms methodology, method, process, and tool are frequently used. These terms are not used alone, but in statements such as ontology development methodology and reuse process. A glossary is provided below to make it easier for readers to understand for what purposes these terms and statements are used:

- **Ontology development methodology:** The term methodology is used only to indicate an ontology development methodology. Ontology development methodologies present the processes and activities that would enable the ontology development to be successful.
- **Ontology reuse process**: The term process is used to refer to the ontology reuse process. Ontology reuse process is sometimes shortly called as reuse process throughout the thesis report. Ontology reuse process is a part of the ontology development methodology. It describes the sub-steps that should be followed during ontology reuse.

- **Ontology modularization method:** The method refers to the ontology modularization method, which extracts smaller and reusable ontology modules from large-scale ontologies.
- **Ontology reuse tool:** The tool defines the ontology reuse tool, a GUI-based Protege plugin that enables the steps of the ontology reuse process and the ontology modularization method to be used by end-users.

Ontology development methodology is the one with the widest scope in these terms. The ontology reuse process is a step in the methodology. Ontology modularization method is a requirement and sub-step in the reuse process. This relation is expressed in the Figure 1.2.

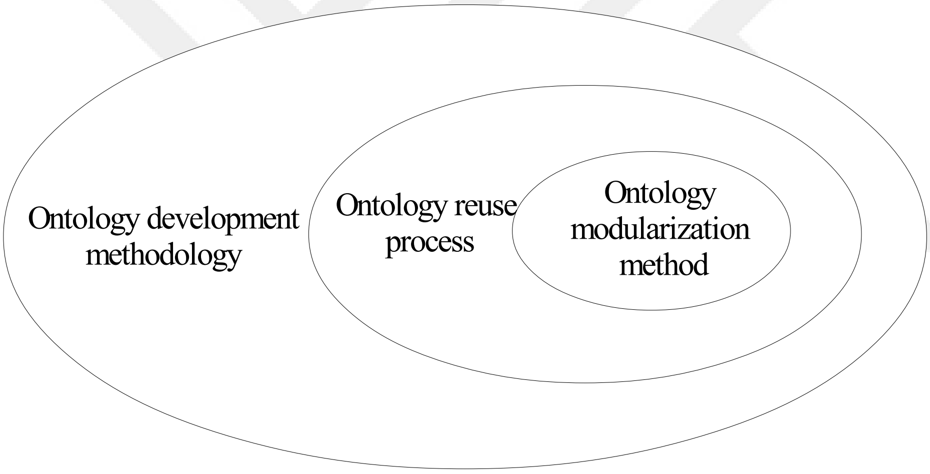

Ontology development methodology
Ontology reuse process
Ontology modularization method

Figure 1.2: Relation between methodology, process and method.

2. BACKGROUND AND THE RELATED WORK

Ontologies are important components in many fields such as knowledge management, natural language processing, machine learning and bio-informatics. They play a key role in the representation of knowledge and interoperability between applications.

Ontology development is a complex task, and a considerable amount of studies have been performed on ontology engineering topics such as methodologies, reuse, integration, evaluation and maintenance. Especially, ontology reuse and integration are still major problems of ontology engineering. Lack of methods and tools that help ontology engineers to reuse existing ontologies, rather than developing ontologies from scratch, increases costs of ontology development process and causes inconsistencies in domain.

Ontology reuse is one of the most challenging parts of the ontology engineering activities. Because it should deal with ontology development methodologies, ontology modularization methods and GUI-based tools to support ontology reuse in an efficient way. From this point of view, this section reviews previous studies on ontologies, ontology development methodologies, ontology reuse processes, ontology modularization methods and ontology tools to understand the state of the art and requirements on ontology reuse.

2.1 Ontology

The term “ontology” originally comes from philosophy. It is a philosophical discipline and deals with nature of being and kinds of existence (Borst, 1997). Ontology defines the categories of beings and their relations. In philosophy, as far as we know, studies about ontology goes back to Aristotle’s Metaphysics[1]. Aristotle defined ontology with his classifications about the world, but the ontology term was first mentioned in 17th century (Øhrstrøm et al., 2005).

AI researchers has adapted ontology into computer science with the emerging requirements about modeling the world. An ontology is a representational artifact which is used for modeling the knowledge of a domain in a structural way, so both

[1] https://plato.stanford.edu/entries/aristotle-metaphysics/

humans and computers could understand concepts of this domain and relationships among them. In the last two decades, especially with Semantic Web vision of Tim Berners Lee (Berners-Lee et al., 2001), ontologies have become far more popular for computer science discipline. Studies on different topics of ontologies, such as ontology learning, ontology mapping, ontology evaluation, ontology development methodologies, ontology reuse, and so on, has soared with the emergence of Semantic Web.

In computer science, the term ontology has defined by many authors with different definitions. Most popular ontology definition in literature has given by Gruber in 1993 (Gruber, 1993):

> “An ontology is an explicit specification of a conceptualization.”

Later, Borst updated this definition with some modifications as follows (Borst, 1997):

> “An ontology is a formal specification of a shared conceptualization.”

Lastly, Studer et al. has blend definitions given by Gruber and Borst (Studer et al., 1998):

> “An ontology is a formal, explicit specification of a shared conceptualization. ”

These definitions describe the term ontology in an academic point of view. Therefore, in order to understand what ontology is clearly in the scope of computer science, some parts of these definitions should be expanded:

- **Conceptualization:** Conceptualization refers to an abstract model of a domain which is both implementation and formal language independent (Uschold and Gruninger, 2004).
- **Explicit:** Every concept, instance and relationship of ontology should be named explicitly (i.e., uniquely and clearly) (Uschold and Gruninger, 2004).

- **Formal:** Abstract model of domain–which is output of conceptualization–should be implemented using a formal modeling language (i.e., ontology language). This process is also known as model specification. Otherwise, conceptual model itself, which is in natural language form or in any other informal notation, could cause ambiguity problems (Uschold and Gruninger, 2004). Relationship between conceptualization and specification is illustrated in Figure 2.1 using model and modeling language (Guizzardi, 2005).

- **Shared:** Ontology is key component of interoperability between humans and applications, so it should be shared and reused by different applications (Uschold and Gruninger, 2004).

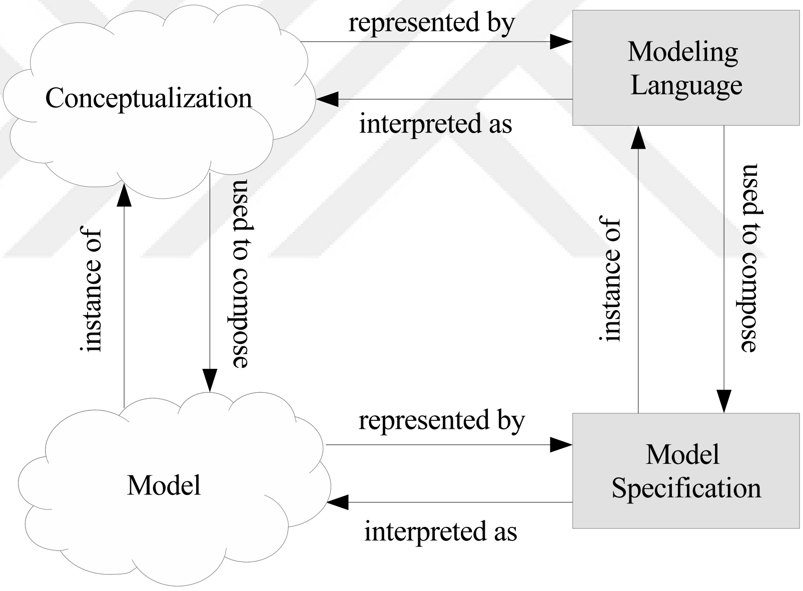


Figure 2.1: Relationship between conceptualization and specification (Guizzardi, 2005).

2.1.1 Types of ontologies

It is possible to classify ontologies from different perspectives such as level of generality, purpose of use and level of formality. Guarino proposed a classification within the level of generality. This classification includes four different types of ontology as seen in Figure 2.2 (Guarino, 1998):

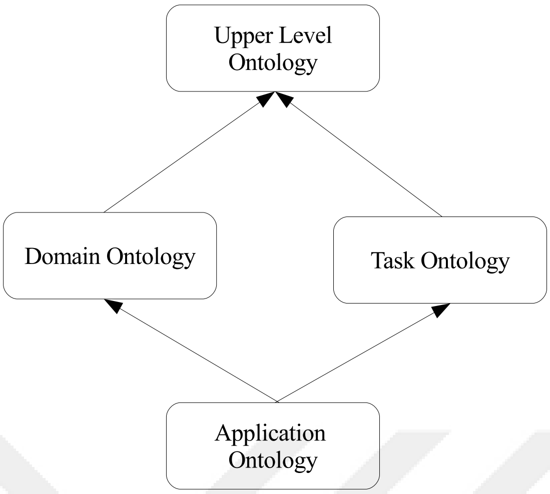


Figure 2.2: Types of ontologies (Guarino, 1998).

- **Upper-level ontology:** Upper-level ontologies, also known as top-level ontologies, define general concepts that are independent of a particular domain or problem. These general concepts consist of basic contents such as time, event, place and person. Basic Formal Ontology (BFO)[2] (Smith et al., 2005), General Formal Ontology (GFO)[3] (Herre, 2010), Dublin-core[4] and FOAF[5] are specific examples of upper-level ontologies.

- **Domain ontology:** Domain ontologies define a model for a general problem domain such as bio-informatics, medicine and e-commerce. They often specialize the definitions of upper-level ontologies. Foundational Model of Anatomy (FMA)[6] (Rosse and Mejino, 2003) and SNOMED-CT[7] (Donnelly, 2006)are examples of general purpose domain ontologies.

- **Task ontology:** Task ontologies describe activities of general a domain, such as diagnosis, treatment, or imaging. (Balakirsky, 2015) and (Freitas et al., 2014) are two examples of OWL-based task and planning ontologies.

- **Application ontology:** Application ontologies are the most specialized

[2] https://basic-formal-ontology.org/
[3] http://www.onto-med.de/ontologies/gfo/
[4] https://dublincore.org/
[5] http://xmlns.com/foaf/spec/
[6] http://sig.biostr.washington.edu/projects/fm/AboutFM.html
[7] http://www.snomed.org/

ontologies compared to other types of ontology. They are specializations of general purpose domain or task ontologies.

In addition, ontologies could also be classified according to their formality level. Uschold and King has proposed four categories for this classification as follows (Uschold and Grüninger, 1996):

- **Highly informal:** Ontology is expressed using natural language. One of the major drawbacks is that natural language can lead to ambiguity problems.
- **Semi-informal:** Ontology is expressed in natural language like highly informal ontologies, but some restrictions and rules are used to eliminate ambiguity and provide a structured form.
- **Semi-formal:** Ontology is expressed in formally defined language.
- **Rigorously formal:** Ontology is expressed with formal semantics, theorems and proofs.

Another topic that should be considered when talking about the types of ontologies is the ontology spectrum. This spectrum emerged from a conversation at the ontology panel at the AAAI 99 conference and was refined by McGuinness to the one stated in Figure 2.3 (McGuinness, 2003). This spectrum refers to the formality and complexity of different semantic models such as dictionary, taxonomy and frames.

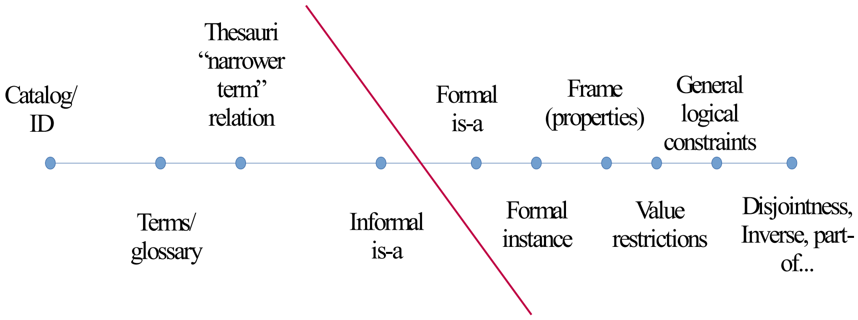


Figure 2.3: Ontology Spectrum (McGuinness, 2003).

2.1.2 Formal languages to build ontologies

Defining ontologies using natural language will certainly lead to ambiguity problems. Therefore, there is a need for structural ontology languages which allow representing knowledge in an explicit and formal way to eliminate these problems. In recent years, several languages have been developed and proposed to build ontologies. While some of these languages are actively used today, others remain only as a reference in the ontology development literature. Some important ontology development languages are briefly discussed below:

- **Knowledge Interchange Format (KIF)** is a formal modeling language to share knowledge between separate applications and developed by Interlingua working group under DARPA as a part of Ontolingua project (Genesereth, 1998). KIF is not intended to be used for knowledge representation. Rather, it is an interchange language as stated in its name, and mostly computer systems that receive information modeled with KIF convert this information into their internal form for later use. KIF is an extension of first-order logic and provides a LISP-like notation (Gruber, 1993) and this makes KIF is a highly expressive language for knowledge representation. But this expressiveness level brings a complexity problem to building ontologies and makes systems heavyweight that have to deal with KIF format (Kalibatiene and Vasilecas, 2011).
- **Resource Description Framework (RDF)**[8] is a standard developed by World Wide Web Consortium (W3C) to describe web metadata (Cyganiak et al., 2014). RDF is a simple language and consists of only three object types: *resources*, *properties* and *statements*. Statements, also called as *triple* , allow to relate properties with resources and consist of three parts: *subject*, *predicate* and *object*. There is no further mechanisms to define relationships between properties and resources (Gomez-Perez and Corcho, 2001). *RDF Schema (RDFS)*[9] is a layer on top of RDF as a semantic extension and provides a data modeling vocabulary for RDF data (Brickley and Guha, 2014). RDFS allows developers to define schema for their RDF model.
- **Ontology Inference Language (OIL)**, which has been developed as part

[8] https://www.w3.org/RDF/

[9] https://www.w3.org/TR/rdf-schema/

of OntoKnowledge project, is a web-based ontology development language based on RDFS (Fensel et al., 2001). While it uses RDF syntax, OIL ontologies are also valid RDF documents. OIL adds frame-based knowledge representation primitives to the RDFS and prevents reification. OIL has been developed using a layered architecture and it consists of four layers. Every layer adds new primitives and complexities to the previous one. Bottom layer is named as *Core OIL* and it includes primitives which have direct relationships with RDFS constructs. Standard OIL adds new semantic primitives to RDFS. Instance OIL allows to define instances of concepts over previous layer. Lastly, Heavy OIL is the last layer of OIL and reserved for future extensions such as rule language (Fensel et al., 2001).

- **DARPA Agent Markup Language (DAML)** program, which is funded by US Government, fund researches in languages, tools and applications to support semantic web development (Mcguinness et al., 2002). As a first step, a new language called DAML+ONT was developed as an extension of RDF (Horrocks, 2002) in this program. At the same time, another group of researchers were developing OIL with similar purposes. Later, both groups merged their efforts and a semantic web language called DAML+OIL[10] arised. While both DAML-ONT and OIL are languages based on RDF, DAML+OIL has also been developed as an extension of RDF. It provides an object-oriented approach to describe the structure of model in terms of classes and properties.

- **Web Ontology Language (OWL)**[11] is a standard ontology development language recommended by W3C (van Harmelen Deborah L. McGuinness, 2004). It's based on RDFS and compatible with earlier ontology development languages, including SHOE and DAML+OIL. It has more power to express semantics and includes primitives such as conjunction, disjunction, universal quantifier and existential quantifier which provide logical inferences that can be used by inference engines (Pulido et al., 2006). OWL has been designed as three sub-languages. OWL Lite is the most restricted subset of the OWL family. It supports classification and simple constraints such as cardinality. While it has advantage of performance and easy implementation because of its lightweight structure, it lacks of semantic expressiveness power. OWL DL supports all OWL constructs, gives maximum expressiveness and retains

[10] https://www.w3.org/TR/daml+oil-reference

[11] https://www.w3.org/OWL/

computational completeness by guaranteeing finite time computations. Because of its correspondence with Description Logics (DL), it is named as OWL DL. Although OWL DL provides maximum expressiveness for ontologies, it has also some restrictions such as type separation. In addition to all the capabilities of OWL DL, OWL Full provides syntactic freedom on RDF but does not guarantee computation time. By using OWL Full, it is possible to augment the meaning of pre-defined RDFS constructs.

2.1.3 TBox vs ABox in the scope of thesis

Since this thesis work is based on OWL ontologies, it is also important to understand the components of ontologies in the context of DL. A typical DL ontology comprise of two parts: *TBox* and *Abox* (De Giacomo and Lenzerini, 1996). *TBox*, where T stands for *terminological*, represents the definitions of concepts and their relationships. In other words, it defines the schema of the ontology. *Abox*, where A is acronym for *assertional*, on the other hand, describes attributes, relationships and roles of the individuals according to *TBox* schema. To better understand what *TBox* and *Abox* is, an example is illustrated in Figure 2.4 using a simple knowledge graph.

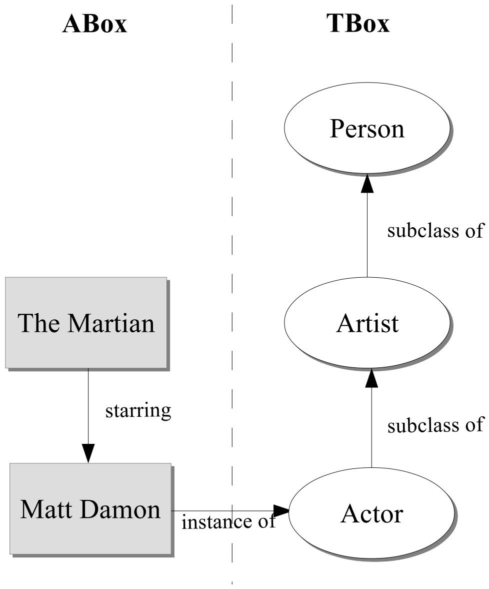


Figure 2.4: TBox vs Abox.

Difference between *TBox* and *Abox* is important for the scope of this thesis project. All the algorithms and methods developed for ontology reuse is specialized according to this distinction.

2.2 Ontology Development Methodologies

With the proliferation of ontologies in computer sciences, studies on ontology development methodologies have increased. The new methodologies, which are generally based on the experiences gained from the developed ontologies, aimed to present the processes and activities that would enable the ontology development to be successful. These studies on processes, methodologies, technologies and tools for ontology development have revealed a new engineering discipline called Ontology Engineering (Suárez-Figueroa et al., 2011).

In this section, important methodologies in ontology development literature are examined. Particular attention has been paid to how these methodologies address ontology reuse. Finally, ontology development methodologies are compared according to life-cycle, reuse and tool support.

2.2.1 Grüninger & Fox

Grüninger and Fox have proposed a methodology based on their experience in the ontologies developed within TOVE project (Grüninger and Fox, 1995). This methodology is one of the first studies in the ontology development literature. They described the steps of the ontology development methodology as follows:

- **Motivating scenarios:** Grüninger and Fox states that, ontology development is motivated by scenarios, which are usually story of problems or examples, that arise in the applications. A motivating scenario should address both problem definition and possible solutions. These motivating scenarios will provide an informal basis for the objects and relationships that will take place in the modeling of the problem domain.
- **Informal competency questions:** These questions depend on the motivational scenarios obtained in the previous step. Since they are not expressed in any formal ontology language, they are called informal competency questions . Each component in the competency questions should

be able to be represented by ontology. Informal competency questions and motivating scenarios provide an informal assessment of the new ontology.

- **Specification in first-order logic-Terminology:** Once informal competency questions are identified, the terminology of the new ontology should be specified using first-order logic.
- **Formal competency questions:** Once the informal competency questions and the terminology of the new ontology have been identified, these competency questions should be formalized. Formalization is performed using first-order logic.
- **Specification in first-order logic-Axioms:** Simply proposing a set of objects does not constitute an ontology. Axioms also should be provided to define the semantics. These axioms define terms and constraints for objects. Axioms should be defined using first-order logic.
- **Completeness theorems:** Once formal qualification questions are defined, the conditions, under which the solutions given to these questions are complete, should be determined.

2.2.2 Uschold & King

Uschold and King have proposed a process framework based on their experience in the development of Enterprise Ontology (Uschold and King, 1995). The main process consists of four steps: identify purpose, building the ontology, evaluation and documentation.

- **Identify purpose:** It is the step where studies are carried out about why ontology is developed and for what purposes it will be used. It is also important to determine the target users and their characteristics.
- **Building the ontology:** This step consists of three sub-steps. *Ontology capture* is the stage of determining the key concepts and relationships in the domain of interest.. After defining the key concepts and relationships, unambiguous texts definitions for such concepts and relationships should be specified. Then the terms that refer to such concepts and relationships should be identified. *Ontology codin*g involves the definition of concepts

and relationships identified in the previous stage with a formal ontology development language. Lastly, building ontology involves *integrating existing ontologies* step. During both of the capture and coding activities, existing ontologies should be reviewed and considered for potential reuse.

- **Evaluation:** A technical assessment of the ontology is performed to understand whether it meets the requirements or not.
- **Documentation**: It is emphasized that there should be a guide for documentation. This guide may differ depending on the type and purpose of ontology.

2.2.3 Methontology

Methontology is a well-structured methodology which enables construction of ontologies at knowledge level and from scratch (Fernández-López et al., 1997). It includes identification of the ontology development, a life-cycle based on evolving prototypes and specific techniques to carry out each activity.

Ontology development methodology includes activities which are carried out during ontology development. Methontology defines three groups for these activities:

- **Project management activities** consist of activities such as planning, control and quality assurance to ensure management throughout the development. *Planning* determines which activities are to be performed, how they will be implemented and resource management. *Control* ensures that events are performed as intended. Quality assurance checks whether quality of each output is sufficient.
- **Development oriented activities** include activities which are directly carried out during ontology development. These activities are specification, conceptualization, formalization and implementation. At the *specification* stage, the purpose, scope, target users, and possible usage scenarios of ontology are determined and described informally. The specification document may also include a list of terms that may be included in the intended ontology. *Conceptualization* includes activities to structure domain

knowledge as meaningful models. In the *formalization* stage, the conceptual model created in the previous stage is transformed into a formal model. Then, *implementation* encodes this formal model using a formal ontology language. Lastly, during the *maintenance* phase, activities are performed to confirm that the ontology is up-to-date and correct.

- **Support activities** are performed simultaneously with development phase. They include knowledge acquisition, evaluation, integration, documentation and configuration management. Most of the *knowledge acquisitio*n is done in the specification phase. Experts, books, figures and existing ontologies are some resources that may be useful for knowledge acquisition. During the *evaluation* phase, a technical assessment is made to determine whether the ontology meets the requirements. Integration involves reuse of existing ontologies. In order to accelerate the development process, already available ontologies related to the domain should be integrated to the new ontology, if possible. *Configuration Management* tracks all versions of outputs such as documentation, software and ontology files. Methontology recommends that, documentation should be performed alongside with all activities.

2.2.4 Ontology Development 101

Ontology Development 101 (Noy and McGuinness, 2001) is a guideline, rather than a complete management methodology, which is based on some object-oriented design principles. Ontology Development 101 includes an iterative life cycle. In each iteration, the steps outlined in the guideline are followed sequentially. The output ontology of each iteration is input to next iteration, and the development continues until all requirements are completed.

Steps of Ontology Development 101 is given below:

- **Determine the domain and scope of the ontology:** It is recommended to start the ontology development process by determining the scope and domain. For this purpose, some simple questions should be answered:
  - What is the domain of the ontology?
  - What is the possible use cases of the ontology?

  - What questions should be answered by the ontology?
  - Who will use the ontology?

  A number of competency questions can also be prepared to determine the scope of ontology.

- **Consider reusing existing ontologies:** Existing ontologies related to problem domain and scope should be evaluated and reused if possible.
- **Enumerate important terms in the ontology:** At this stage, it is aimed to enumerate the important terms related to the domain of ontology. Overlap between concepts, relationships among concepts and possible properties of concepts should not be taken into consideration. Also, it is not important to determine details such as whether a term is class or property.
- **Define the classes and the class hierarchy:** The class hierarchy can be determined using a top-down, bottom-up, or hybrid approach.
- **Define the properties of classes:** Determining classes and hierarchical relationships between them is not sufficient for developing an ontology. Also, internal structure of concepts should be described by defining the properties of these concepts.
- **Define the facets of the slots:** Properties have facets such as value type, values it can take, and the cardinality. These facets should be described to complete property definition.
- **Create instances:** At this stage, instances of each class in the hierarchy are created.

2.2.5 DILIGENT

The DILIGENT (Pinto et al., 2006) methodology states that ontology engineering should be distributed, loosely-controlled and evolving. This approach includes some key roles responsible for development.

In the DILIGENT approach, an initial version of core ontology is made available to all users and users are free to make any changes they want in their local workspace. There is a control board who is responsible for the quality of core ontology, and this authority checks every committed change performed by distributed users.

DILIGENT consists of five main steps:

- **Build:** Field experts, information engineers, users and ontology engineers work together to create the initial core ontology.
- **Local adaptation:** Once the core ontology is available, each user is free to make changes in their local workspace. However, they are not authorized to directly merge these changes to the core ontology. The board is responsible for controlling every commit.
- **Analysis:** Board analyzes local ontologies and requests to find similarities in user ontologies. This task is critical, since not every local change can take place in the next core ontology version.
- **Revise:** Board should periodically review the shared ontology. This prevents local ontologies from diverging too far from shared ontology.
- **Update:** Once a new version of the shared ontology is released, users should update their local workspace.

2.2.6 Neon

Neon methodology (Suárez-Figueroa et al., 2012), unlike other ontology development methodologies, does not have strictly defined steps and flow. Instead, 9 different scenarios have been proposed that can be used for different development requirements.

Neon is a methodology focused on the reuse of existing ontologies and resources. It consists of 4 main components:

- **Neon glossary:** This glossary defines the processes and activities (see 2.1) that can take place in various steps of ontology development.
- **Scenarios:** Each scenario consists of required processes and activities of Neon glossary. 9 different scenarios have been proposed to provide guidelines

Table 2.1: Processes and activities defined in Neon glossary (Suarez-Figueroa et al., 2012).

Processes			
Ontology aligning	Non-ontological resource reuse	Ontology module reuse	Ontology reuse
Ontology design pattern reuse	Ontological resource reuse	Ontology re-engineering	Ontology statement reuse
Activities			
Ontology annotation	Ontology merging	Ontology evolution	Ontology restructuring
Ontology assessment	Ontology modification	Ontology extension	Ontology reverse engineering
Ontology comparison	Ontology modularization	Ontology feasibility study	Ontology feasibility study
Ontology conceptualization	Ontology module extraction	Ontology formalization	Ontology search
Ontology configuration management control	Ontology partitioning	Ontology forward engineering	Ontology selection
Ontology customization	Ontology population	Ontology implementation	Ontology specialization
Ontology diagnosis	Ontology pruning	Ontology integration	Ontology summarization
Ontology documentation	Ontology quality assurance	Knowledge acquisition for ontologies	Ontology translation
Ontology elicitation	Ontology repair	Ontology learning	Ontology update
Ontology enrichment	Ontology requirements specification	Ontology localization	Ontology upgrade
Ontology environment study	Non-ontological resource reverse engineering	Ontology mapping	Ontology verification
Ontology evaluation	Non-ontological resource transformation	Ontology matching	

for common situations. The name of each scenario clearly indicates the purpose of this scenario and the activities and processes to be implemented within it. These scenarios are listed below:

  - Scenario 1: From specification to implementation
  - Scenario 2: Reusing and re-engineering non-ontological resources
  - Scenario 3: Reusing ontological resources
  - Scenario 4: Reusing and re-engineering ontological resources
  - Scenario 5: Reusing and merging ontological resources
  - Scenario 6: Reusing, merging, and re-engineering ontological resources
  - Scenario 7: Reusing ontology design patterns (ODPs)
  - Scenario 8: Restructuring ontological resources
  - Scenario 9: Localizing ontological resources

- **Life-cycle:** Neon does not strictly indicate which life-cycle model should be used. Instead, it makes recommendations for which model is most suitable for particular conditions. The developer can choose from waterfall and iterative models.

- **Guidelines for processes and activities:** The guidelines include recommendations for processes and activities.

2.2.7 UPON Lite

UPON (Unified Process for Ontology Building) Lite (De Nicola and Missikoff, 2016) is a lightweight, simple and agile ontology engineering methodology. Primary purpose of UPON is creating a simple and easy-to-use methodology to reduce the role of ontology engineers during development. Thus, most of the development responsibility can be shifted to end users such as domain experts and non-technical stakeholders. It is also recommended to use simple tools such as document processors and spreadsheets instead of semantic technology tools that could make the development difficult for ordinary users.

UPON recommends 6 consecutive steps for ontology development. The output of each step constitutes the input to the next step. These steps are described below:

- **Terminological level:** The first step of UPON methodology is creating a domain specific terminology. This terminology includes list of terms that characterize the problem domain.

- **Glossary level:** Not all terms in the terminology, created in previous step, may have a clear and widely accepted meaning. Therefore, every term in the terminology should be enriched with a textual explanation. Beside descriptions, terms should also be divided into conceptual categories. UPON Lite adopts an ontology structuring method, OPAL. This method groups the concepts into three main categories–object, process and actor–and three auxiliary categories–complex, atomic and reference. Lastly, in glossary level step, synonym concepts should be revealed.

- **Taxonomy Level:** In order to establish taxonomy, the list of terms obtained in the first two steps are organized hierarchically in this step. It is not enough to determine *ISA* organization between existing terms. Users also should introduce more abstract and generic terms to complete hierarchy.

- **Predication Level:** To complete the entity definitions, properties of each entities should be described in this step. Users identify atomic properties (simple data fields such as *price* and *date*), complex properties (properties that have internal structure such as *address*) and reference properties (properties that refer to other entities).

- **Parthood Level:** This step focus on part-of relations between terms.

- **Ontology Level:** Ontology engineers use the semi-formal outputs of previous steps and build the ontology using that knowledge.

2.2.8 SAMOD

SAMOD (A Simplified Agile Methodology for Ontology Development) is an agile methodology for ontology development, which is inspired by agile software engineering principles and test-driven development (Peroni, 2017). SAMOD focuses on simplicity like all other agile development methodologies. It involves domain experts and ontology engineers along ontology development.

SAMOD organizes the ontology development as three simple steps that are repeated within iteration cycles. These steps are detailed below:

- **Define a new test case:** Ontology engineers and domain experts work together to write down informal motivating scenarios. They also should produce informal competency questions using these motivating scenarios. Then, motivation scenarios and competency questions are used to identify the terms that may be involved in the problem domain to build a glossary. The rest of this step is led by ontology engineers. Ontology engineers should develop a modelet using motivating scenarios, competency questions and glossary terms. Modelet refers to the sub-model defined for the conceptualization of a motivation scenario. It is created independently of other scenarios and is not referenced by another model. SAMOD recommends some principles to develop modelet:

  - Keep it small
  - Use patterns
  - Middle-out development
  - Keep it simple
  - Self-explanatory entities

  This modelet should developed using a graph language to easily convert it to OWL, later. Ontology engineers should test this modelet to proceed. Using motivation scenarios, competency questions and glossary terms, ontology engineers create test cases. These test cases includes example *ABox* datasets and formal SPARQL queries. To proceed to next step, these test cases should be executed successfully. An example of competency question is shown is Table 2.3.

Table 2.3: An example competency question.

Identifier	5
Question	What are the possible types of vehicle class?
Answer	All vehicles used in land, sea and air transport
Samples	Car, ship, train
Related Questions	1, 2

- **Merge the current model with modelet:** The current model, which is output of previous iteration, should be merged with the modelet which is created in this iteration, by ontology engineers.

- **Refactor the current model:** Most important part of refactoring is reusing existing knowledge. Also, for a successful refactoring, documentation and technological enhancements should be considered.

2.2.9 Comparison of ontology development methodologies

The main objective of this thesis is to develop an innovative process for ontology reuse that aims to solve existing problems (see sections 2.3.2 and 2.3.3). In order to achieve this goal, it is necessary to understand the requirements, current studies and awareness on ontology reuse. One of the most important sources in which the current situation can be observed is ontology development methodologies. How these methodologies handle and position ontology reuse in their steps is important for the results of this thesis.

In this section, ontology development methodologies are examined within the scope of ontology reuse. The awareness of these methodologies on ontology reuse and the proposed solutions to current problems are elaborated. As a result, it is observed that awareness on ontology reuse and its requirements is quite high, but there is not enough tool and process support to solve the existing problems.

Many of the ontology development methodologies include ontology reuse as a separate step. In some methodologies, reuse takes place as a step of main development while others use it as a supporting process during all other steps. Although it is stated that ontology reuse is very important in terms of both development costs and interoperability, details are usually not included.

Ontology development methodologies are compared according to some reuse criteria determined within the scope of this study. Meaning of each criteria is summarized below:

- **Life-cycle model:** Specifies which life-cycle model the methodology is based on (waterfall, iterative, prototyping, etc.).

- **Ontology reuse:** Indicates whether ontology reuse is included in the methodology.
- **Details of ontology reuse:** Specifies whether ontology reuse is enlisted as a step without any recommendation and directive, or it is detailed as a process in itself.
- **Position of ontology reuse:** Indicates whether ontology reuse is a major step in the main process or a supporting step that is used by all other steps.
- **Tool/plugin support:** Specifies whether a tool or plugin support is available for ontology development and reuse principles stated by methodology. Although such a requirement is not expected directly from any ontology development methodology, it is important to provide such a support to facilitate the process.

The methodologies examined in this section are compared in Table 2.4 according to the criteria summarized above.

2.3 Ontology Reuse

Reuse is a key factor for the success of ontology development. Besides it is a means to increase quality and decrease development costs, reusing of ontologies also enables and enhances interoperability among applications, by eliminating inconsistencies (Simperl, 2009). When an application reuses an ontology developed by another application, interoperability between these applications results naturally without the need for mapping and integration (Katsumi and Grüninger, 2017).

Reuse is one of the complex tasks of ontology development, with many benefits, but with additional costs. These additional costs prevents engineers from reuse activities, and usually causes ontologies to be developed from scratch. Therefore, to reduce these costs and encourage ontology engineers, it is essential to support ontology reuse process with specialized methods and tools.

Table 2.4: Comparison of ontology development methodologies.

Methodology	**Life-cycle model**	**Ontology reuse**	**Details of ontology reuse**	**Location of ontology reuse**	**Tool/plugin support**
Grüninger & Fox	Not specified	Not specified	Not specified	Not specified	Not specified
Uschold & King	Not specified	Specified	Undetailed	Separate step	On-to-knowledge
Methontology	Evolving prototypes	Specified	Undetailed	Supporting step	ODE
Ontology Development 101	Iterative	Specified	Undetailed	Separate step	Not specified
Diligent	Iterative	Not specified	Not specified	Not specified	Not specified
Neon	Optional (iterative or waterfall)	Specified	Detailed	All methodology is based on reuse	Neon Toolkit
UPON Lite	Iterative	Not specified	Not specified	Not specified	Not specified
SAMOD	Iterative	Specified	Undetailed	Separate step	Not specified

2.3.1 Reusability

Reusability, as stated in its definition, is one of the key features of ontologies. Ontologies are inherently reusable components, and can be directly used by other ontologies without any additional requirements. However, some additional features will increase the likelihood of reuse of ontologies and facilitate the reuse process. Although it is not easy to list these features, some studies have been carried out to determine the features that are considered necessary, and thus to increase reusability.

Wilkinson and colleagues (Wilkinson and et al, 2016) have proposed the FAIR principles to increase the reusability of data. FAIR stands for *Findability*, *Accessibility*, *Interoperability* and *Reusability*, which are detailed below:

- **Findability:** First step of reuse is finding data, so metadata and data should be easily found by human and computers through a registry or index.
- **Accessibility:** Once data has been found, it should be accessed using standard communication protocols such as HTTP, FTP or STMP. Used protocol should allow to authentication/authorization, if necessary.
- **Interoperability:** Data should be defined using a broadly-accepted language for knowledge representation. Also, reused data should conform with FAIR principles, if possible.
- **Reusability:** Final goal of FAIR principles is optimizing the reuse of data. Data should be richly described, released with a clear and comprehensible license, associated with detailed provenance and meet domain-relevant community standards.

Tim Berners-Lee's 5-star open data principles (Berners-Lee, 2015), which defines the levels of being linked and open, also give some clues about requirements of reusability. Each star level also indicates the level of being reusable, implicitly. These stars are summarized below:

- **1-star:** Data should be available through web using an open license.

- **2-star:** Data should be defined using a structured format (e.g. Excel instead of image of a table).
- **3-star:** Data should be defined using a non-proprietary format (e.g. CSV instead of Excel's internal data format).
- **4-star:** Unique URIs should be used to denote things.
- **5-star:** Existing linked data should be reused.

2.3.2 Current state of ontology reuse

While many efforts have been performed to help and facilitate the reuse of ontologies, because of the difficulty of the problem, it is still an open issue for ontology development community (Katsumi and Grüninger, 2016). In recent years, some studies have been carried out on the rate of ontology reuse to see the final state of the problem.

One of these studies has been performed on OBO (Open Biological and Biomedical Ontology) Foundry ontologies. The reuse and overlap rates have been examined on snapshots taken in 3 different time periods, and results shown in Table 2.5 have been obtained (Ghazvinian et al., 2011).

Table 2.5: Reuse and overlap rates of OBO Foundry for different time periods (Ghazvinian et al., 2011).

	September 2009	**March 2010**	**September 2010**
Total terms	272,168	311,351	318,872
Reused terms	10,972 (4.0%)	13,458 (4.3%)	17,067 (5.4%)
Overlap terms	9,598 (3.5%)	9,566 (3.1%)	9,992 (3.1%)

Although the data shown in Table 2.5 covers a short period, it gives clues about the reuse rates and number of overlapped terms in OBO Foundry. From these values, it can be seen that the reuse rate is around 4-5%. Although there is a small increase in the reuse rate over time, the number of overlapped terms is also increasing. The authors state that the reuse rate of 5% is still low, and lack of semi-automated methodologies and tools causes this problem (Ghazvinian et al., 2011).

Another research to analyze reuse has been carried out by Poveda et al. (2012) on ontologies included in Linked Open Data Vocabulary[12] (LOV) (Vandenbussche et al., 2017). They observed that 40.53% of the terms are reused, which is a quite high rate. However, further experiments shows that 67% of reused elements are directly imported from other ontologies using *owl:import*. Even if a single term is reused from an imported ontology, other terms are also imported with the reused one. Therefore, the rate of reuse appears to be high. It is also possible that importing ontologies can cause performans problems during ontology processing.

Lastly, one of the important research about reuse rates has been performed by Kamdar et al. (2016) on Bioportal (Whetzel et al., 2011)[13] ontologies. Results show that reuse is around 3% and 4% for different types of reuse approaches, while term overlap is 14.4%.

Although the number of studies summarized above is low, they give some clues about the shortcomings of ontology reuse, and it is possible to summarize these shortcomings as follows:

- Reuse rates are quite low (~5%) even in controlled and domain-specific ontologies, such as OBO Foundry and Bioportal.
- Lack of reuse increases overlap rates, because locally defining an existing term causes duplicates. Ghazvinian et al. (2011) shows that number of overlapped terms is increasing over time. Also, rate of overlapped terms in Bioportal is 14.4% (Kamdar et al., 2017).
- There are no semi-automated processes and tools to support reuse (Bontas et al., 2005) (Ghazvinian et al., 2011) (Kamdar et al., 2017). Lack of processes and tools that can help ontology engineers increase the costs of reuse processes and cause solutions such as *owl:import* to be preferred.
- Ontology modularization is important to support reuse (Ghazvinian et al., 2011) (Kamdar et al., 2017). Modularization is required

[12] https://lov.linkeddata.es/dataset/lov/
[13] https://bioportal.bioontology.org/

because large-scale ontologies cannot be imported directly. However, the lack of modularization methods that can be integrated with the reuse process restricts reuse.

- Study of Poveda et al. (2012) shows that preferred way of ontology reuse is still *owl:import*.
- *Reuse is still a problem for ontology development.*

2.3.3 What limits ontology reuse?

Ontology reuse is an important factor for the success of ontologies and ontology development. However, the current state of reuse is still not at the desired level, due to number of reasons. Obrst et al. (2014), have given details about these reasons:

- **Mismatch and misunderstanding:** Ontology engineers usually believe that existing ontologies do not fit into their requirements. This is usually due to the fact that, users and engineers attempt to find an ontology for potential reuse, before the requirements of the newly developed ontology is not clearly defined. First, domain and competency questions of new ontology must be understood. Once the requirements are clearly defined, one can delve into existing ontology resources, to find a suitable ontology.
- **Finding right ontology:** Another obstacle that arises during ontology development is that, sometimes it can be hard to find ontology resources to search ontologies. Efforts such as Bioportal, Linked Open Data Vocabulary (LOV) and Open Ontology Repository (OOR)[14] are currently addressing this problem. But, these repositories indexing ontologies with their basic metadata. Also, detailed information such as competency questions, mappings and alignments with other ontologies, ontological commitments and design decisions should be included with ontology. Thus, it can be easier to determine the right ontology. In addition, this metadata can be a guide to understand which parts of ontology should be reused.
- **Ontology does not fit:** Existing ontologies, that satisfy the requirements of new ontology, may have some issues that prevent them being reused by

[14] http://oor.net/

other ontologies. For example, an ontology may be incomplete or developed using a knowledge representation language which is not appropriate. Such situations can make it difficult to reuse ontology.

- **Modularity:** Some ontologies, which fit to requirements of newly developed ontology, may include hundreds of thousands of concepts and millions of relations. In such cases, users should be able to use specific parts of the ontology, using modularization techniques. Ontology modularization should become a part of ontology development. Also, ontology repositories may provide support about modularization to make it easier to reuse huge ontologies.
- **Integration:** Integration of multiple ontologies is a requirement that may arise during reuse. Ontology mapping is a useful technique that should not be ignored for this kind of integration problems.
- **Development from scratch:** Sometimes, users believe that, it may be easier to develop ontology from scratch, instead of spending efforts to find the right ontology to reuse. Users can overcome this problem using processes and tools that support ontology reuse.
- **Localization:** Many ontologies are designed in English. Although it is logical for the ontology developer to use his or her own language, identifiers should not include language specific terms. Instead, localization should be provided using labels.

Both the problems mentioned in Section 2.3.2 and the obstacles listed in Section 2.3.3 must be partially or completely resolved in order to increase ontology reuse. These items are summarized in Table 2.6 and potential solutions are given.

In the following sections of the thesis, it is explained how the studies have been carried out to solve these problems. Also, Section 3.3.5 summarizes which problems are solved.

2.3.4 How to reuse ontologies?

The simplest way of reuse is performed with import. Most of the knowledge representation languages provide constructs, e.g. *owl:import*, to directly import

Table 2.6: Problems of ontology reuse and possible solutions.

Problem	**Possible solution**
Low reuse rates	A reuse-based and iterative ontology development methodology and ontology reuse tools
Mismatch and misunderstanding	A reuse-based and iterative ontology development methodology
Finding right ontology	Searching capabilities integrated with ontology reuse tools
Ontology does not fit	Ontology evaluation techniques
Modularity	Modularization methods integrated with ontology reuse tools
Integration	Ontology reuse tools that supports integration
Development from scratch	A reuse-based and iterative ontology development methodology and ontology reuse tools

existing ontologies, and nearly all of the ontology development environments, such as Protege, support these constructs. Import is usually suitable for small ontologies, such as FOAF, BFO and GFO, because existing ontology is entirely imported into new ontology. Ontology development tools load all elements of imported ontologies.

Although importing ontologies as a whole is suitable for small ontologies, it is not applicable to large and comprehensive ontologies, including many concepts and relationships such as Dbpedia (Auer et al., 2007)[15], Yago (Suchanek et al., 2008)[16], SNOMED-CT and FMA. One possible way of reusing from bigger ontologies is including only required terms. This can be achieved in two ways (Kamdar et al., 2017): Concept Unique Identifier (CUI) and Internationalised Resource Identifier (IRI). CUIs are mappings for same terms defined in different ontologies. For example, Unified Medical Language System (UMLS) uses CUIs to map terms with similar meaning in different terminologies. In IRI reuse, concept is defined exactly in one ontology, and reused by other ontologies using its unique identifier. Figure 2.5 shows example of both reuse types (Kamdar et al., 2017).

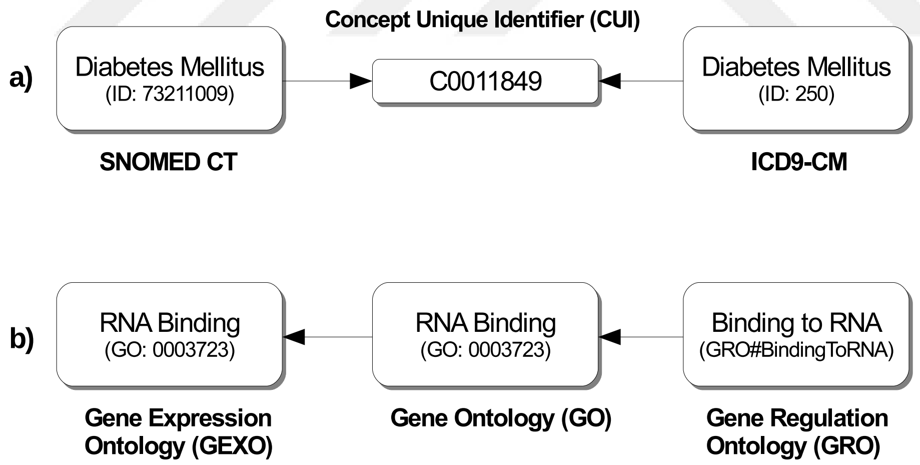

Concept Unique Identifier (CUI)
a)
Diabetes Mellitus (ID: 73211009)
C0011849
Diabetes Mellitus (ID: 250)
SNOMED CT
ICD9-CM
b)
RNA Binding (GO: 0003723)
RNA Binding (GO: 0003723)
Binding to RNA (GRO#BindingToRNA)
Gene Expression Ontology (GEXO)
Gene Ontology (GO)
Gene Regulation Ontology (GRO)

Figure 2.5: Types of reuse: a) CUI reuse, b) IRI reuse (Kamdar et. al, 2016).

Another alternative to reuse from bigger ontologies is modularization (Kamdar et al., 2017). Users and engineers can extract modules (sub-ontologies) from large ontologies using this approach. Modularization is a difficult problem to implement without tools and method support. Many efforts have been performed to

[15] http://wikidata.dbpedia.org/

[16] https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/

support ontology modularization and section 2.5 gives detailed information about these studies.

2.3.5 Where to find ontologies?

With the increasing number of ontologies, resources are needed to provide access to these ontologies and make it easier for users to search and use ontologies. These resources are also important to enhance ontology reuse, providing findability and accessibility for existing ontologies. Ontology resources can be divided into three categories: search engines, libraries and standalone resources:

- **Ontology search engines** crawl the Web to index ontologies (d'Aquin and Noy, 2012). They do not collect or store them. Swoogle (Ding et al., 2004), Watson (Sabou et al., 2007) and OntoSearch (Zhang et al., 2005) are examples of ontology search engines.
- **Ontology libraries or repositories** are web-based systems that allows to store and access ontologies. These systems usually contain similar ontologies for a particular scope or domain. They also provide searching mechanisms to find ontologies. For example; Bioportal, OBO Foundry and OLS[17] (Ontology Lookup Service) are important libraries for biomedical ontologies.
- **Standalone resources** are generally large ontologies such as Dbpedia, Yago and Wikidata[18] (Vrandečiundefined, 2012). These ontologies are stored in their own server. They usually provides SPARQL endpoints to allow users to search.

2.4 Ontology Reuse Tools

Attempting to reuse ontologies without any tool support is a costly and difficult process, which also discourages users on ontology reuse. Therefore, in ontology reuse literature, there are tools which are developed to facilitate reuse of ontologies.

[17] https://www.ebi.ac.uk/ols/index

[18] https://www.wikidata.org/

In this section, existing tools and plugins developed for ontology reuse is introduced. Also, these tools are compared at the end of the section (see Section 2.4.6).

2.4.1 DOG4DAG

DOG4DAG (the Dresden Ontology Generator for Directed Acyclic Graphs) is a semi-automated tool that allows creating and extending ontologies using the text in web, PubMed and PDF repositories (Wächter and Schroeder, 2010). DOG4DAG works on English text to extract terms. It works like a ontology learning tool using natural language and statistical techniques, rather than an ontology reuse tool.

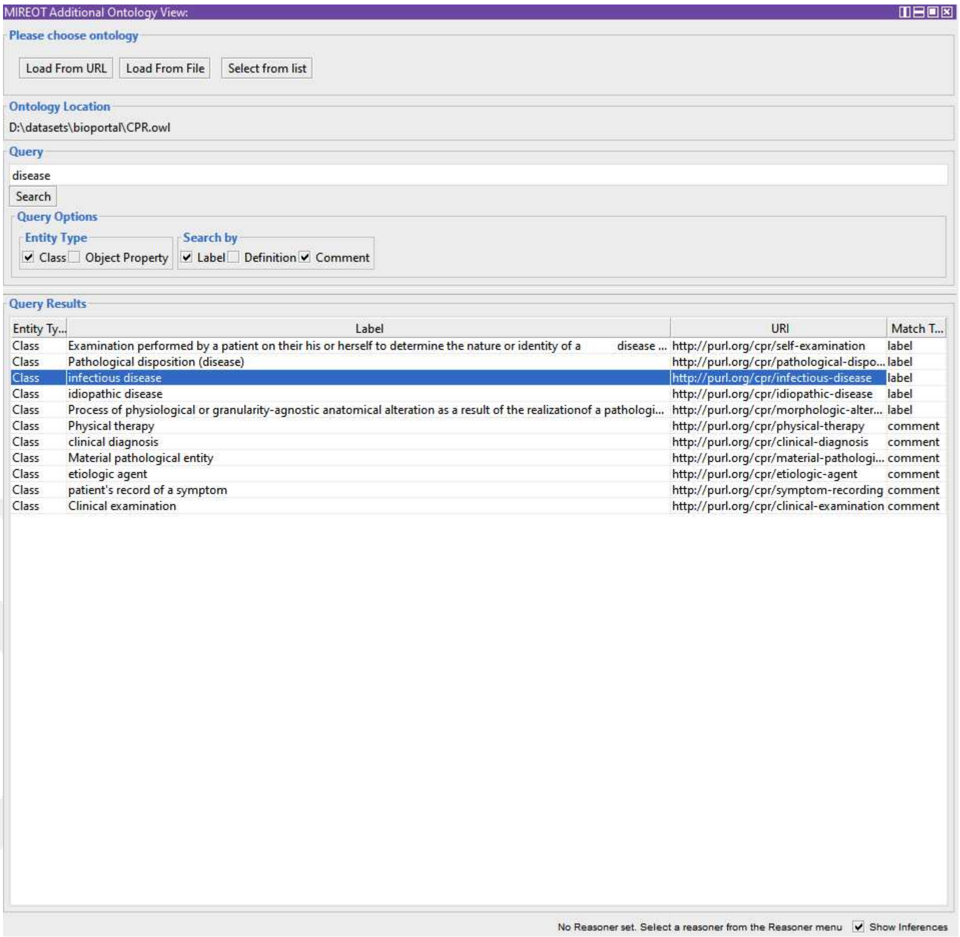
DOG4DAG was developed as OntoEdit and Protege plugins, but binaries for Protege plugin is not available anymore and project is not active.

2.4.2 MIREOT

MIREOT (Minimum Information to Reference an External Ontology Term), is a Protege plugin that supports importing terms from external ontologies using MIREOT principles (Hanna et al., 2012). These principles states that, following minimal information is sufficient to reference an external term consistently:

- URI of source ontology
- URI of referenced term
- URI of direct super-classes
- Labels and textual definitions to ease development (not required)

MIREOT plugin uses these principles and import selected term from external ontology. Screenshot taken from Protege plugin is shown in Figure 2.6. As can be seen on the figure, users can get reused ontology via URL or local file. Users can also select from pre-defined ontologies. Then the type of entity (class or property) to be searched is selected and search text is entered. Selected concepts are integrated into ontology according to MIREOT principles.



MIREOT Additional Ontology View:

Please choose ontology

Ontology Location

D:\datasets\bioportal\CPR.owl

Query

disease

Query Options

Entity Type: ☑ Class □ Object Property

Search by: ☑ Label □ Definition ☑ Comment

Query Results

Entity Ty...	Label	URI	Match T...
Class	Examination performed by a patient on their his or herself to determine the nature or identity of a disease ...	http://purl.org/cpr/self-examination	label
Class	Pathological disposition (disease)	http://purl.org/cpr/pathological-dispo...	label
Class	infectious disease	http://purl.org/cpr/infectious-disease	label
Class	idiopathic disease	http://purl.org/cpr/idiopathic-disease	label
Class	Process of physiological or granularity-agnostic anatomical alteration as a result of the realizationof a pathologi...	http://purl.org/cpr/morphologic-alter...	label
Class	Physical therapy	http://purl.org/cpr/physical-therapy	comment
Class	clinical diagnosis	http://purl.org/cpr/clinical-diagnosis	comment
Class	Material pathological entity	http://purl.org/cpr/material-pathologi...	comment
Class	etiologic agent	http://purl.org/cpr/etiologic-agent	comment
Class	patient's record of a symptom	http://purl.org/cpr/symptom-recording	comment
Class	Clinical examination	http://purl.org/cpr/clinical-examination	comment

No Reasoner set. Select a reasoner from the Reasoner menu ☑ Show Inferences

Figure 2.6: MIREOT Protege plugin.

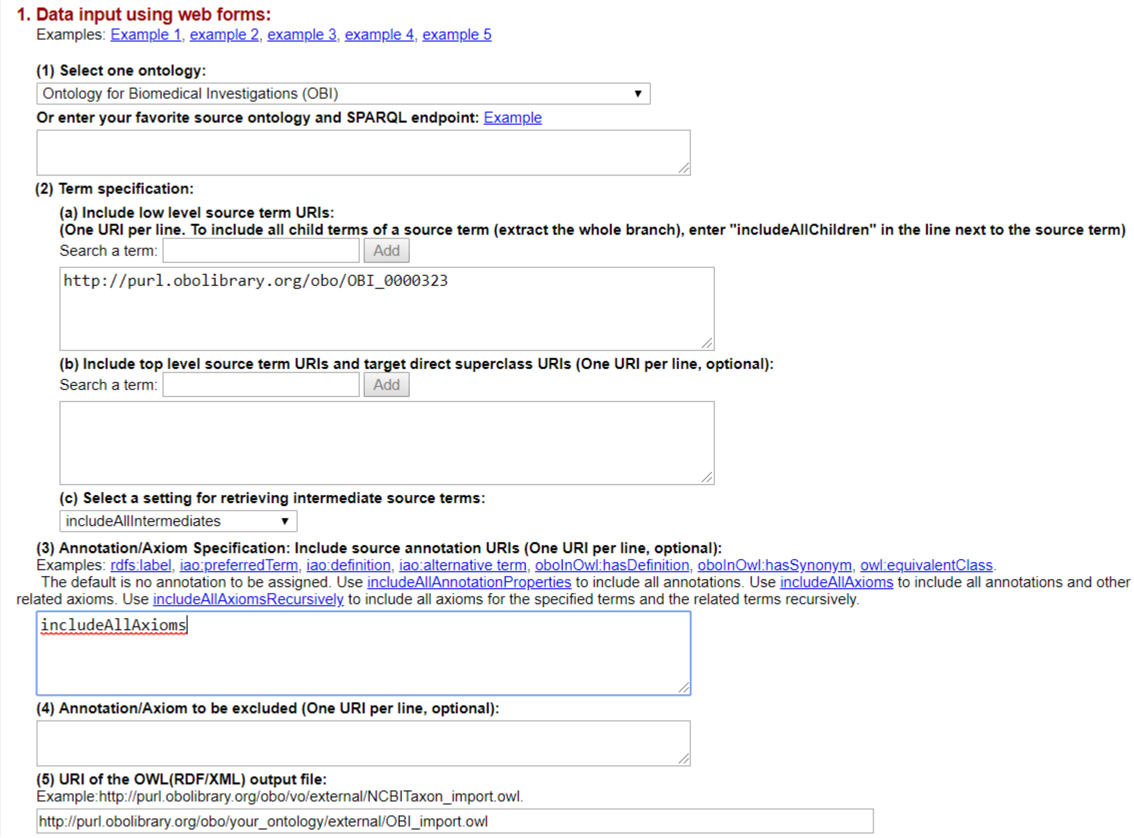
2.4.3 OntoFox

OntoFox[19] is a web-based platform that enables reusing terms and axioms from external ontologies (Xiang et al., 2010). OntoFox implements MIREOT principles and facilitates the development of ontologies by automatically creating modules around the selected terms. These modules are serialized as RDF/XML and could be downloaded to directly import.

OntoFox has some pre-loaded OBO ontologies. Users can use these ontologies or state a specific SPARQL endpoint to reuse terms. Parameters such as *includeAllAxioms* and *includeAllAxiomsRecursively* can be used to specify the details of module to be extracted.

[19] http://ontofox.hegroup.org/

Screenshot of OntoFox web page is shown in Figure 2.7. As can be seen on the figure, user can select from pre-defined ontologies or enter SPARQL endpoint for other ontologies. Then user enters low-level source term URIs and top-level source term URIs. If *includeAllIntermediates* parameter is active, OntoFox includes paths between top-level and low-level terms. Users can also specify excluded axioms using a separate input box.



1. Data input using web forms:
Examples: Example 1, example 2, example 3, example 4, example 5

(1) Select one ontology:
Ontology for Biomedical Investigations (OBI) ▾
Or enter your favorite source ontology and SPARQL endpoint: Example

(2) Term specification:
(a) Include low level source term URIs:
(One URI per line. To include all child terms of a source term (extract the whole branch), enter "includeAllChildren" in the line next to the source term)
Search a term: Add

```
http://purl.obolibrary.org/obo/OBI_0000323
```

(b) Include top level source term URIs and target direct superclass URIs (One URI per line, optional):
Search a term: Add

(c) Select a setting for retrieving intermediate source terms:
includeAllIntermediates ▾

(3) Annotation/Axiom Specification: Include source annotation URIs (One URI per line, optional):
Examples: rdfs:label, iao:preferredTerm, iao:definition, iao:alternative term, oboInOwl:hasDefinition, oboInOwl:hasSynonym, owl:equivalentClass.
The default is no annotation to be assigned. Use includeAllAnnotationProperties to include all annotations. Use includeAllAxioms to include all annotations and other related axioms. Use includeAllAxiomsRecursively to include all axioms for the specified terms and the related terms recursively.

```
includeAllAxioms
```

(4) Annotation/Axiom to be excluded (One URI per line, optional):

(5) URI of the OWL(RDF/XML) output file:
Example:http://purl.obolibrary.org/obo/vo/external/NCBITaxon_import.owl.
http://purl.obolibrary.org/obo/your_ontology/external/OBI_import.owl

Figure 2.7: OntoFox web page.

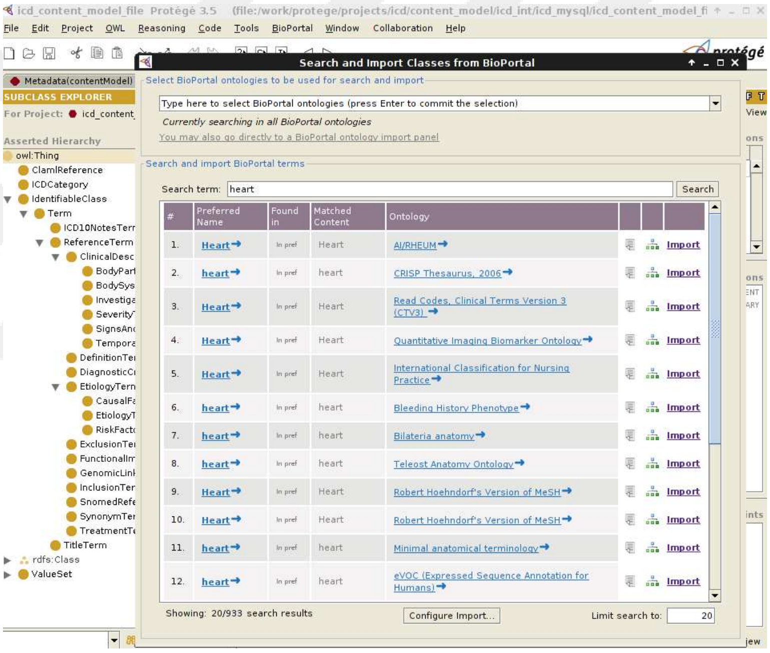
2.4.4 Bioportal Import Plugin

Bioportal Import Plugin is a Protege plugin that allows importing terms from Bioportal ontologies (Nair et al., 2011). It has a generic and configurable structure for different requirements. It supports the following features for ontology reuse:

- Import single class or sub-tree of class hierarchy, up to intended level
- Import into ontology which is currently opened in Protege editor, or into another ontology

- Import labels and other textual definitions
- Import metadata for classes or ontologies
- Save current configuration for later reuse

This plugin does not work anymore because of updated Bioportal APIs as of May 2014. Therefore, the screenshots shown in Figure 2.8, is taken from official website of plugin (Nair and Tudorache, 2011). The figure shows how terms can be searched in Bioportal ontologies using Bioportal Import plugin. Selected concepts can be imported into the ontology individually or as a sub-tree with its all parents.



Search and Import Classes from BioPortal

Select BioPortal ontologies to be used for search and import

Type here to select BioPortal ontologies (press Enter to commit the selection)

Currently searching in all BioPortal ontologies

You may also go directly to a BioPortal ontology import panel

Search and import BioPortal terms

Search term: heart [Search]

#	Preferred Name	Found in	Matched Content	Ontology			
1.	Heart	In pref	Heart	AI/RHEUM			Import
2.	heart	In pref	heart	CRISP Thesaurus, 2006			Import
3.	Heart	In pref	Heart	Read Codes, Clinical Terms Version 3 (CTV3)			Import
4.	Heart	In pref	Heart	Quantitative Imaging Biomarker Ontology			Import
5.	Heart	In pref	Heart	International Classification for Nursing Practice			Import
6.	heart	In pref	heart	Bleeding History Phenotype			Import
7.	heart	In pref	heart	Bilateria anatomy			Import
8.	heart	In pref	heart	Teleost Anatomy Ontology			Import
9.	Heart	In pref	Heart	Robert Hoehndorf's Version of MeSH			Import
10.	Heart	In pref	Heart	Robert Hoehndorf's Version of MeSH			Import
11.	heart	In pref	heart	Minimal anatomical terminology			Import
12.	heart	In pref	heart	eVOC (Expressed Sequence Annotation for Humans)			Import

Showing: 20/933 search results [Configure Import...] Limit search to: 20

Figure 2.8: Bioportal Import Plugin (Nair and Tudorache, 2011).

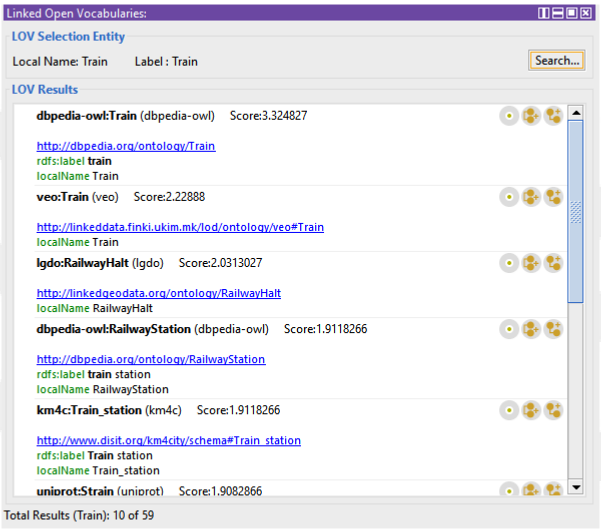
2.4.5 ProtegeLOV

ProtegeLOV is an open source tool that allows importing terms from LOV (Linked Open Data Vocabularies) to ease ontology development (García-Santa et al., 2015). Protege LOV supports following features for ontology reuse:

- Search for terms in LOV repository
- Reuse terms directly

- Add term and define equivalent class (or equivalent property) axiom
- Add term and define sub-class (or sub-property) axiom

Plugin screenshot is shown in Figure 2.9. As shown in figure keyword “Train” is searched in LOV repository. Results are listed with scores. User can select from three different reuse strategy: reuse directly, add with equivalent class axiom or add with sub-class axiom.



Local Name: Train Label : Train

dbpedia-owl:Train (dbpedia-owl) Score:3.324827
http://dbpedia.org/ontology/Train
rdfs:label **train**
localName Train

veo:Train (veo) Score:2.22888
http://linkeddata.finki.ukim.mk/lod/ontology/veo#Train
localName Train

lgdo:RailwayHalt (lgdo) Score:2.0313027
http://linkedgeodata.org/ontology/RailwayHalt
localName RailwayHalt

dbpedia-owl:RailwayStation (dbpedia-owl) Score:1.9118266
http://dbpedia.org/ontology/RailwayStation
rdfs:label **train** station
localName RailwayStation

km4c:Train_station (km4c) Score:1.9118266
http://www.disit.org/km4city/schema#Train_station
rdfs:label **Train** station
localName Train_station

uniprot:Strain (uniprot) Score:1.9082866

Total Results (Train): 10 of 59

Figure 2.9: ProtegeLov plugin (Garcia-Santa et al., 2015).

2.4.6 Comparison of ontology reuse tools

Ontology reuse tools are one of the most important components that support ontology reuse. These tools aim to eliminate the cost and development challenges of reuse, by automating standard and recurring activities. However, the features and capabilities of these tools are very limited. In addition, many of them are not active and not supported anymore.

In this section, the main features of ontology reuse tools are compared to each other to understand the current status better. Although DOG4DAG may be used for reuse in some scenarios, it is unfair to include it in the comparison. Because, it is mostly used for ontology learning purposes. Therefore, the comparison has been made for the other four tools (See Table 2.7).

As shown in the comparison, half of the reuse tools are not active anymore.

Bioportal Import Plugin does not provide plugin binaries and ProtegeLov does not work due to API changes. In such a case, it may not make sense to compare these tools. However, to reveal the current situation and understand the shortcomings, it is important to make this comparison in order.

The common preference for ontology reuse tools is being developed as a Protege plugin. Only exception, which is examined in this report, is OntoFox. OntoFox has been developed as a standalone web page. It is possible to talk about pros and cons for both choices. However, even though a web interface is offered, it is important to facilitate the work of ontology engineers, providing a plug-in for Protege, which has become the de-facto standard in ontology development.

Providing a Web API (SOAP, RESTFUL, etc..) is an important factor for making these tools reusable. In this way, ontology engineers and developers can also integrate these services with their own tools for specific purposes and make ontology reuse even more widespread. The tools Bioportal Import Plugin, OntoFox and ProtegeLov, which are mentioned in this study, provide Web APIs and run their own extensions through these interfaces. On the other hand, MIREOT works locally and does not have any Web interface support.

Some of the ontology reuse tools keep ontologies in their own databases and provide access through their services. For example, Bioportal Import Plugin allow access to Bioportal ontologies. On the other hand, Ontofox has built-in support for OBO repositories.

Ontology reuse tools usually provide graphical user interfaces to interact with users. These interfaces are generally form-based pages and provides basic functionalities such as search, list, select and import. However, it is also important to provide graph-based interfaces that can clearly demonstrate the relationships between concepts and enable navigation. Unfortunately, none of these tools supports such an interface.

Another significant disadvantage of these methods is that they do not provide a true solution for ontology modularization and do not integrated with a reuse process. Usually, selected concept(s) and its definition are imported into ontology using MIREOT and similar rules.

Table 2.7: Comparision of ontology reuse tools.

	Status	**Type**	**Web API**	**Built-in Database**	**Graph-based GUI Support**	**Ontology Module Support**	**Methodology support**	**Extracting class definition**
MIREOT	Active	Plugin	No	No	No	No	No	MIREOT Principles
OntoFox	Active	Standalone	Yes	Yes	No	Yes	No	MIREOT Principles
Bioportal Import Plugin	Not Active	Plugin	Yes	Yes	No	No	No	MIREOT Principles
Protege Lov	Not Active	Plugin	Yes	Yes	No	No	No	No

2.5 Ontology Modularization Methods

The reuse of small and modular ontologies is relatively easy process. But sometimes, it is required to reuse knowledge from large and well-accepted domain ontologies, such as SNOMED-CT and FMA. During reuse of these ontologies, there may be a need to extract smaller parts from the original ontology, that satisfy the needs of users, rather than importing the ontology as a whole or trying to reuse the related concepts individually. This process is called as *ontology modularization* (Doran, 2009) and one of the most important factors for ontology reuse (Kamdar et al., 2017).

Ontology modularization process can be defined formally as follows: Let us to simply define an ontology as a pair of concepts and relations $O = (C, R)$, where *C* stands for concepts (both classes and properties) and *R* is relations connecting classes and properties to each other. Ontology module $M = (C^m, R^m)$, which is output of ontology modularization, is a subset of original ontology *O*, where $M \subseteq O$,$C^m \subseteq C$ and $R^m \subseteq R$ (Doran, 2009).

Modularization can support many ontology engineering problems beyond reuse. Spaccapietra et al. (2005) summarizes these problems as follows:

- **Scalability:** It's very hard to process large ontologies for different purposes such as reasoning, information retrieval and visualization.
- **Complexity management:** The larger the ontology developed, the more complex is the design, development and maintenance processes.
- **Understandability:** It is easier to understand ontology, if it is designed as smaller modules.
- **Personalization:** Modular ontologies have more advantageous for organizing ownership. Ownership information can be attached to each module independently.
- **Reuse:** Reusing smaller ontologies is easier than reusing larger ones.

Recently, several approaches have been proposed for carrying out ontology modularization problem. All of these approaches aim to simplify reusability and management of huge ontologies. In general, all of these methods may be grouped into two main categories (Doran, 2009):

- **Partitioning:** These methods focus on dividing ontologies into smaller and more reusable partitions. These methods are also named as network partitioning.
- **Module extraction (Sub-ontology extraction):** These methods extract a sub-ontology from original ontology. The extraction algorithms make the module extraction according to selected ontology elements and user preferences. These selected elements are mostly classes from original ontology and provided by end users.

Partitioning is a more statistical approach than module extraction. Generally, algorithm runs over the ontology and ontology modules are generated. Doran et. al (2007), define these methods as "one-shot approaches". The extracted partitions could be efficient for reusing and especially for managing of huge ontologies, but users need sometimes could go beyond of these partitions. For instance, user could expect a sub-ontology containing two concepts which are located in different partitions. In circumstances like previous example, module extraction can provide a more flexible method. Also, module extraction can be used in an iterative and incremental reuse approach more efficiently, and is more suitable to the need for a semi-automatic reuse process and tool, which is also stated by Kamdar et al. (2016).

To build an iterative and incremental reuse process and modularization method, this study focuses on module extraction (sub-ontology extraction) methods rather than partitioning. Therefore, in the following sections, existing studies on module extraction methods in the literature are detailed and compared.

2.5.1 Noy & Musen

One of the earliest and important attempts at module extraction problem has been carried out by Noy and Musen (2004). This approach has been developed to work on RDFS ontologies. Although, it is also possible to adapt it to OWL ontologies with simple modifications.

Noy and Musen call the extracted ontology modules as *ontology views* similar to database views. To create ontology views, one should define *Traversal Specification*s. These specifications include initial concept list and *Traversal Directives* to expand ontology view around these concepts. Then, modularization algorithm builds *Traversal Views* using the given definitions.

To understand this approach better, some definitions that forms the traversal specifications should be detailed:

- **Concept definition:** If a concept or instance C is to be included in ontology module, the following rules should be executed for its definition $Def(C)$:
  - All RDF statements in which C is the subject should be included in ontology module.
  - All properties P in which C is the domain should be included in ontology module. Also, if C is a instance, all triples that defines property values v for $< C, P, v >$ should be included.

  This rule is first applied to the initial concepts chosen by the user. Then, it is repeated recursively for all new concepts added to the sub-ontology.

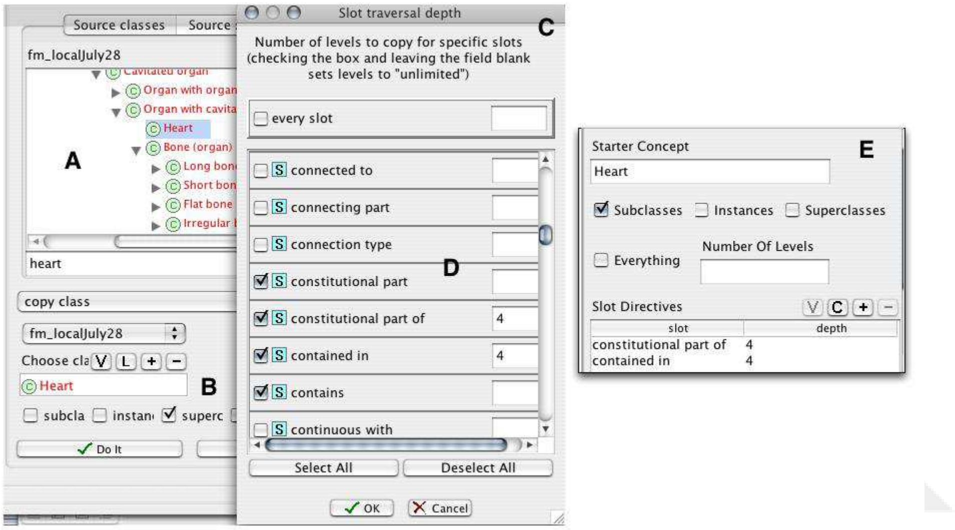
- **Traversal Directive:** A traversal directive $D = (C_{st}, PT)$ is a pair, where C_{st} is the initial concept of traversal, and PT is a set of property directives (P, n). where P is a property, and n is a positive integer (or infinity) that defines the length of the traversal along the P. This rule specifies how to traverse, starting from concept C_{st}. For example, suppose that, user has selected *ClinicalFinding* class as initial concept C_{st}, and *findingSite* property as P. Then, modularization algorithm can traverse ontology starting from *ClinicalFinding* using *findingSite* property.

- **Traversal View Specification:** A traversal view specification is a set of TD (traversal directives).

- **Traversal Directive Result:** Traversal directive result $D(O)$ is the result of running traversal directive D on ontology O. It refers to ontology view resulting from the execution of a traversal directive.

- **Traversal View:** A traversal view $TV(O, T)$ is union of all traversal directive results, where O is original ontology, and T is traversal view specification.

This method also has a Protege plugin. Because plugin executables are not accessible anymore, it is not possible to run and test the plugin. A screenshot taken from original paper is shown in Figure 2.10 which illustrates the steps of the method. User first selects the initial concepts (A), and specifies traversal directives for built-in RDFS properties (B). Then, a seperate window (C) should be opened to define traversal directives for user defined properties of ontology (D). Lastly, user can review the definitions using another simple form (E).



fm_localJuly28

Heart

□ subcla □ instan ☑ superc

Number of levels to copy for specific slots (checking the box and leaving the field blank sets levels to "unlimited")

□ every slot

□ S connected to

□ S connecting part

□ S connection type

☑ S constitutional part

☑ S constitutional part of 4

☑ S contained in 4

☑ S contains

□ S continuous with

Starter Concept: Heart

☑ Subclasses □ Instances □ Superclasses

□ Everything

Number Of Levels

slot	depth
constitutional part of	4
contained in	4

Figure 2.10: Noy and Musen: Protege plugin (Noy and Musen, 2004).

2.5.2 Seidenberg

Seidenberg (2009) has proposed a module extraction method, which is based on traversing ontology graph using predefined rules and user's choice. This method has been optimized to work on GALEN ontology which has been produced as part of the OpenGALEN project (Rector et al., 2003). GALEN ontology does not use the full expressivity of OWL-DL. Instead, it is based on the $SHIF$[20] subset. So, the algorihtm is constrained to these rules. But it is possible to apply the core of the approach to other ontologies.

The algorithm executes the following steps to extract an ontology module:

- The algorithm takes one or more user selected classes as input to extract an ontology module.

[20] OWL-Lite is based on $SHIF$ description logics and is less expressive than OWL-DL.

- Users can also filter properties that should not be traversed. Property filtering is only applicable for GALEN ontology, because it is based on particular property types.
- All the superclasses of each input class are added to ontology module recursively. It is particularly avoided to merge superclasses to reduce the level of hierarchy. Because, authors state that, it could destroy the semantic accuracy of the hierarchy.
- Subclasses of each input class are traversed recursively and added to ontology module. But, it is not allowed to go downwards while traversing the superclasses. Otherwise,it is possible to extract entire hierarchy as ontology module.
- While traversing the hierarchy of each target class, equivalent classes, unions, intersections and restrictions should be considered. All these definitions need to be included in the module.
- Superclasses and subclasses of newly included classes and properties are also included recursively.
- Users can specify boundary classes and depth limits to limit the module size.

2.5.3 Materialized Ontology View Extractor (MOVE)

MOVE is another effort on ontology module extraction introduced by Bhatt and his colleagues (Bhatt et al., 2006). MOVE is a traversal approach which is based on extraction rules and user preferences. Because of the costs of extraction process on huge ontologies, algorithm is designed to be distributed. The module extraction algorithm has three phases as illustrated in Figure 2.11.

Import layer (1) performs the transformation of specific ontology representations to the MOVE's internal format, which is compliant with the extraction process. Thus, different ontology formalizations, such as RDF, RDFS, OWL and DAML+OIL, can be supported.

Labeling (2) is the step in which the user selects which components–classes and properties–must be included in the ontology module and which components must not. Labeling is critical and very important to accurately determine the scope and size of the module. This step is performed at the beginning of the module extraction process by users. Intermediate steps may also re-apply labeling to provide communication between different components of algorithm.

Extraction process (3) is the most important part of the MOVE algorithm. This step includes four optimization schemes that handle various issues during module extraction. These optimization schemes are detailed below:

- **Requirements Consistency Optimizations Scheme (RCOS):** RCOS ensures the consistency of the user requirements which are specified during the labeling phase. It contains four sub-schemes:
  - RCOS1: If a binary relation is selected to be included in the ontology module, the two classes which are connected by this relation can not be deselected.
  - RCOS2: If an attribute mapping (i.e. domain/range relation) is selected, the attribute and the class that are associated by this mapping must be selected to be included in the ontology module.
  - RCOS3: If an attribute mapping is deselected, the attribute which is one side of the mapping must also be deselected.
  - RCOS4: If an attribute is selected, but a concept which is connected to this attribute with an attribute mapping is not selected, then there must be a valid path between this attribute and another concept. Attribute can not be included as an isolated entity.
- **Semantic Completeness Optimization Scheme (SCOS):** SCOS rules guarantee the semantic completeness of the extracted ontology module. SCOS includes three sub-schemes:
  - SCOS1: If a class is selected, all its superclasses and inheritance relations must be selected recursively.
  - SCOS2: If a class is selected, all part-of classes and aggregation relations must be selected.
  - SCOS3: If a class is selected, all attributes of this class, with minimum cardinality greater than zero, and their attribute mapping must be selected.

- **Well Formedness Optimization Scheme (WFOS)**: Although all user-specified requirements (labeling) are consistent, these requirements can lead to inconsistencies which causes ontology module to be not valid. WFOS prevents these inconsistencies from happening. It contains five sub-schemes.
- **Total Simplicity Optimization Scheme (TSOS)**: TSOS rules ensures that resulting ontology module is valid and is the smallest possible solution. It contains three sub-schemes.

Optimization schemes are at the core of the extraction process. MOVE algorithm run these schemes in a distributed manner to improve performance. Distribution is the primary focus of this modularization method.

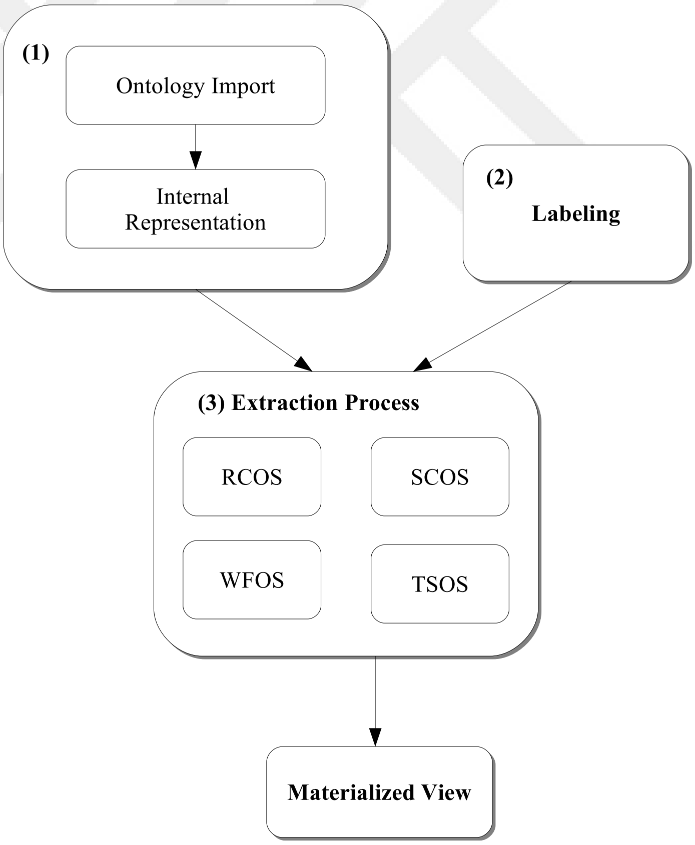


Figure 2.11: MOVE: Sequential extraction process (Bhatt et al., 2006).

2.5.4 Doran et al.

Doran and his colleagues define an ontology module extraction method primarily for reuse (Doran et al., 2007). This approach brings a difference when compared to others, because it is presented as a step of a proposed reuse methodology. The steps of this methodology is illustrated in Figure 2.12 and detailed.

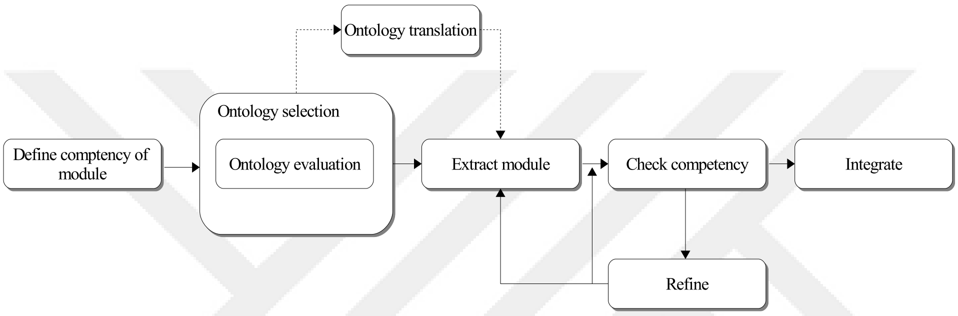


Figure 2.12: Modular reuse architecture (Doran et al., 2007).

- **Defining competency of ontology module:** One should define the competency questions for ontology module before module extraction. These questions give details about what ontology module should express, and what the requirements are.

- **Ontology selection:** User needs to select an ontology which is relevant to their needs. This selection should be based on competency questions. Also, ontology evaluation techniques may be used to review possible ontologies.

- **Ontology translation:** To make the selected ontology compatible with extraction process, it may be necessary to translate ontology from its original representation language to internal format. While making the transformation, to avoid breaking the semantics, differences between expressiveness power of languages should be taken into consideration.

- **Module extraction:** Module extraction is performed on the selected ontology according to user requirements and predefined rules.

- **Check competency**: The ontology module should be checked against competency questions to verify that they are met.

- **Refine:** If competency questions are not met, user should refine the ontology module.

- **Integrate:** If the ontology module is valid against the competency questions, user can integrate it into the ontology being developed.

Module extraction is the most important part of this methodology, and its implementation has more details than the others. The pseudo-code of the module extraction algorithm is given in Algorithm 2.1. Time complexity of this algorithm is $O(n^2)$.

Assuming that competency questions are defined and ontology selection is completed, before module extraction, selected ontology should be translated to an abstract graph model $G = (V, E)$, where V are vertices, and E are edges of this graph. The graph is directed and every edge is labeled to preserve semantics.

Algorithm 2.1 Module extraction algorithm (Doran et al., 2007).

Input: Ontology as a directed graph $G = (V, E)$, a starting vertex s, list of excluded edges $Excluded$, list of laredy visited vertices $Visited$, list of vertices to visit $ToVisit$

Output: Ontology module $M = (V_M, E_M)$

```
function EXTRACTMODULE(s)
    if $s \notin Visited$ then
        insert $s$ into $Visited$
        create a list $C = \{e \in E | s \times \sum_E \times v\}$
        while C is not empty do
            $e \leftarrow$ get first item of $C$
            if $e \notin Excluded$ then
                $e \cup E_M$
                insert all $t$ such that $e = s \times \sum_E \times t$ into $ToVisit$
            end if
            if $ToVisit$ is not empty then
                $tv \leftarrow$ get first item of $ToVisit$
                delete $tv$ from $ToVisit$
                $extractModule(tv)$
            else
                return $M$
            end if
        end while
    end if
end function
```

This method takes only a single concept as an input and extract ontology module around this concept. In the first iteration, *disjointWith* relations are discarded. Because, being disjoint with a class requires that these two classes can not have any instances in common. According to authors, if concept s is selected as starting concept, then user is probably is not interested in its disjoint classes. But, for next iterations, *disjointWith* should be taken into consideration. To prevent to extract an ontology module which is nearly as big as original ontology, they also do not allow navigating to upwards from subclasses of starting concept.

Using the rules defined above, Algorithm 2.1 runs on selected ontology using graph traversals recursively, and extract an ontology module. The module extraction method has been implemented as a standalone and GUI-based modularization tool. This tool is named as *ModTool*, and use Jena (McBride, 2002) to work on ontologies. Also, to check consistency of ontologies, Pellet (Sirin et al., 2007) is used by the application.

2.5.5 Miao et al.

Miao et al. has proposed a simple and graph-based method which handles ontology module extraction on RDF and RDFS ontologies (Miao et al., 2008). This approach includes two separate algorithms for both RDF and RDFS ontologies. The first algorithm is simple and works on RDF ontologies using graph abstraction. Second algorithm reuse the first one and make further improvements for RDFS ontologies.

In order to understand the whole method, firstly, RDF algorithm should be detailed. An RDF graph can defined as $G = (N, E)$, where N is set of nodes, and E is set of edges. Ontology module can simply be extracted deleting all concepts other than user selected ones N_1, and edges which are not incident to N_1. The RDF extraction algorithm is given in Algorithm 2.2.

As shown in Algorithm 2.2, the method proposed by Miao and his colleagues is very simple and only focus on removing concepts and their relations, which are not selected by user. Authors state that, this extraction algorithm can not run RDFS ontologies, without considering inference rules. Otherwise, some of the derived statements, which should be included in the ontology module, can be deleted. To

Algorithm 2.2 RDF module extraction algorithm (Miao et al., 2008).

```
Input: RDF graph $G = (N, E)$, $N_1$ is the set of user selected concepts, $k$ is the
    number of nodes in $N$
Output: Extracted RDF module $G_1 = (N_1, E_1)$
$N_2 = N - N_1$
for $i = 1tok$ do
    if $n_i \in N_2$ then
        $N = N - n_i$
        for $j = 1 \to degree(n_i)$ do
            $E = E - e_j$
        end for
    end if
end for
$E_1 = E$
$N_1 = N$
$G_1 = (N_1, E_1)$
```

prevent this, firstly, an algorithm that computes the closure of RDFS ontology using inference rules is executed. This algorithm run in a distributed manner to improve performance.

From the authors point of view, abstraction of RDFS ontologies as a graph may lead to inconsistencies. Figure 2.13, which includes a part from SNOMED-CT, illustrates this inconsistency. On the left site, *Causative agent*, which connects two concepts, represents an edge. But, on the right side, the same property represents a node. Miao et al. consider this situation as an inconsistency, and handle RDFS graph as two parts: the main body and the restricted body. Main body includes classes and relations which only connect these classes. On the other hand, the restricted body includes attributes and their relations.

Closure algorithm runs on the RDFS ontology and computes the closure of graph $c(G)$. Then, this closure is divided into two parts: the closure of main body $c_1(G)$, and the closure of restricted body $c_2(G)$. Lastly, the extraction algorithm given in Algorithm 2.2 is executed on $c_1(G)$ to extract a module. This RDFS extraction algorithm is given in Algorithm 2.3.

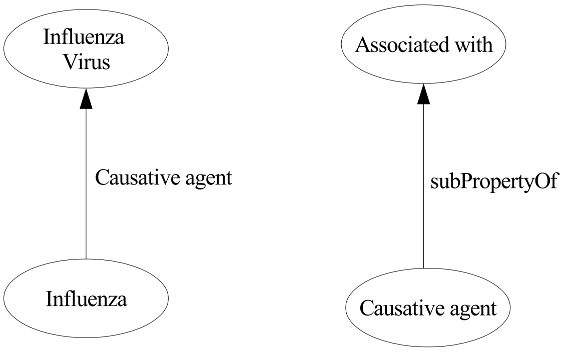


Figure 2.13: Inconsistency in RDFS graph (Miao et al., 2008).

Algorithm 2.3 RDFS extraction algorithm (Miao et al., 2008).

```
Input: RDFS graph $G = (N, E)$, $N_1$ is the set of user selected concepts, $k$ is the number of nodes in $N$
Output: Extracted RDFS module $G_1 = (N_1, E_1)$
1: apply inference rules, and generate closure $c(G) = (N', E')$
2: divide the closure into closure of the main body $c_1(G)$, and closure of the restricted body $c_2(G)$, where $c(G) = c_1(G) \cup c_2(G)$
3: execute RDF extraction algorithm on $c_1(G)$
```

2.5.6 SOMET

Doran and his colleagues have proposed an ontology module extraction framework, which is named as SOMET, to implement different ontology module extraction techniques within a single and common framework (Doran et al., 2008). Thus, all these techniques can work together, and users can adapt and apply multiple methods to extract modules according to their needs.

SOMET is not a new approach for ontology module extraction. Rather, it is an RDF and SPARQL based framework, where different ontology module extraction methods can be implemented. Users can use their own methods implementing SPARQL queries, as well as built-in methods within the framework. Core algorithm of SOMET framework, which performs the module extraction, is given in Algorithm 2.4. The parameters of algorithm are original ontology which is represented as a RDF graph, a signature which describes the module and usually specified by user, and SPARQL queries of selected module extraction method. Algorithm executes SPARQL queries for each visited concept and append query results to the ontology module.

Algorithm 2.4 SOMET: module extraction algorithm.

```
Input: Ontology $O = (V, E)$ which is represented as an RDF graph, a signature $S$
    where $S \subseteq V \cap E$, a set of SPARQL queries $Q$
Output: Ontology module $M$
function MODULEEXTRACTION(O, S, Q)
    $Visited \leftarrow$ initialize visited nodes as an empty set
    $ToVisit \leftarrow$ initialize nodes to visited as an empty set
    insert $S$ into $ToVisit$
    while $ToVisit$ is not empty do
        $n \leftarrow$ get first item of $ToVisit$
        delete $n$ from $ToVisit$
        if $n \in Visited$ then
            for all $q$ in $Q$ do
                $qr \leftarrow$ execute $q$ on $O$
                insert subjects in $qr$ into $ToVisit$
                $M \leftarrow M \cup r$
            end for
        end if
    end while
    return $M$
end function
```

SOMET framework is implemented in Java using Jena API, and Pellet is used as reasoner. Architecture of framework is illustrated in Figure 2.14.

2.5.7 Hussain & Abidi

Hussain and Abidi have proposed another rule-based method for ontology module extraction. They focus on the weaknesses of methods proposed by Bhatt et al. (2006) and Miao et al. (2008). They summarize these weaknesses as follows:

- In MOVE approach, it is possible to extract entire graph as a module due to the SCOS rules, if there is a valid path between every node pair. Even if the whole ontology is not extracted as a module, the resulting module can include very general concepts that the user is not interested in.

- In Miao et al. (2008) approach only user selected concepts and their relations are appended to extracted module. Hence, the module is very restricted and does not contain enough information of user's interest. User need to have in-depth knowledge about original ontology to extract an ontology module

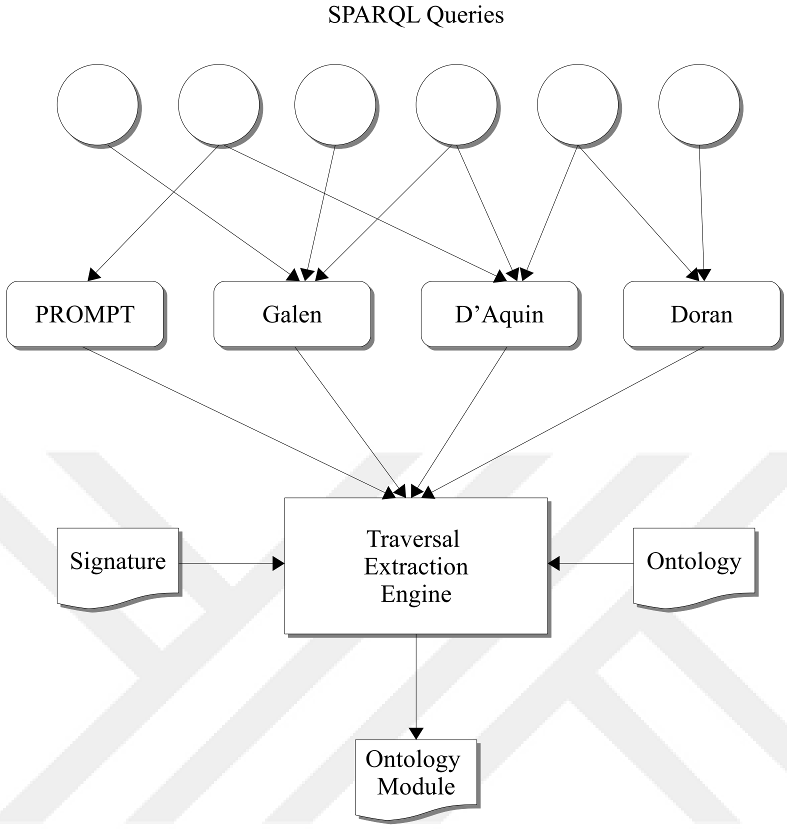


Figure 2.14: SOMET framework (Doran et al., 2008).

which is sufficient for requirements.

In order to solve these problems, Hussain and Abidi present an approach that extract a module which is rich enough, between Bhatt et al. and Miao et al. When compared to MOVE, one of the key features of this study is that, superclasses of a class are not recursively included in the module. But, to preserve the semantic completeness, all roles of the superclasses are inherited and defined in the original class.

Module extraction rules of the algorithm is listed below[21]:

- Include user selected concepts (input)
- Include instances of selected concepts

[21] All module extraction rules are defined as N3 statements

- Include roles of selected concepts
- Include all subconcepts of selected concepts
- Include all roles whose domain concepts are labeled as selected
- Include all concepts which are defined as range for extracted roles
- Include all roles of superconcepts (Superconcepts are not extracted recursively, instead roles of these concepts appended to the selected concept)
- Include intersection and union definitions of extracted concepts (These definitions are reduced to *subClassOf* definitions)
- Include equivalent classes of extracted concepts
- Include complement of extracted concepts
- Include restrictions of extracted concepts
- Include equivalent properties of extracted properties
- Include inverse properties of extracted properties

In order to give an idea about the N3 statement rules, which are executed during extraction process, some of these rules are listed in Table 2.8.

Table 2.8: N3 rules executed by ontology module extraction algorithm (Hussain and Abidi, 2009).

Rule name	**N3 statement**
Subconcepts	*{?O usr:selected ?C. ?C a rdfs:Class. ?S rdfs:subClassOf ?C} => {?O usr:selected ?S}*
Equivalent classes	*{?O sbont:extract*[22] *?C. ?C owl:equivalentClass ?S.} => {?O sbont:extract ?S}*
Roles	*{?O usr:selected ?C. ?C a rdfs:Class. ?P rdfs:domain ?C} => {?O sbont:extract ?P}*
Intersection	*{?C owl:intersectionOf ?L. ?L a rdf:List. ?X list:in ?L} => {?C rdfs:subClassOf ?X}*
Union	*{?C owl:unionOf ?L. ?L a rdf:List. ?X list:in ?L} => {?X rdfs:subClassOf ?C}*

[22] *sbont:extract* is a function that returns the list of items already included in the ontology module

2.5.8 Ranwez et al.

One of the approaches based on graph theory has been proposed by Ranwez et al. (2012). Their study abstracts ontology as a *Direct Acyclic Graph (dag)*, and make use of two common graph operators: *least common ancestor (lca*) and *greatest common descendants (gcd)*. Core of their method only works on class hierarchy and omits other semantic relations such as equivalent class, domain, range, etc., during graph traversal.

lca operator computes the least common ancestors, which are the closest and common superclasses of selected nodes, using the traversal algorithm. As all the other module extraction methods, this method also starts with user concept selection. With the set of user selected concepts S, $lca(S)$ finds the nearest ancestors of these classes. As illustrated in Figure 2.15, the result of lca operator may contain multiple concepts (least common ancestors of selected classes $C1$, $C2$ and $C3$ are $A1$ and $A2$).

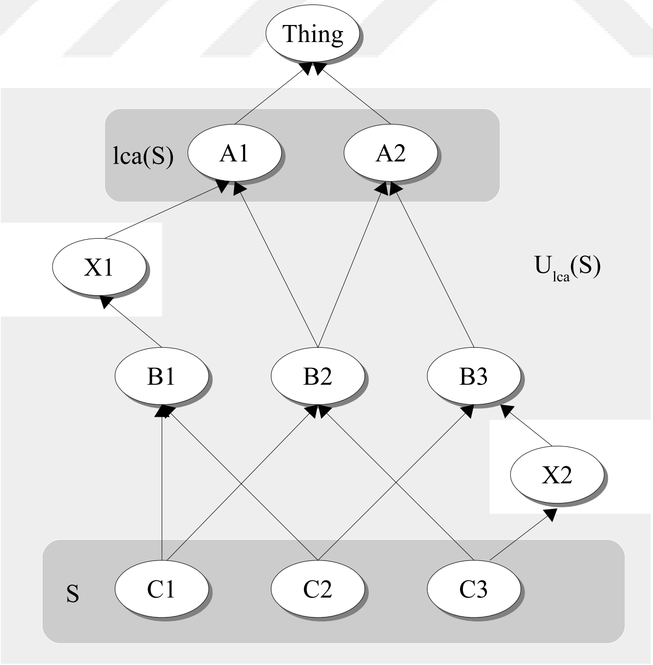


Figure 2.15: An example of lca and U_{lca} operators (Ranwez et al., 2012).

lca operator only finds the common ancestors of selected classes. Ranwez et al. also defines another operator $U_{lca}(S)$, to build the class hierarchy of selected concepts. This operator unions all least common ancestors of all possible pairs of selected classes. Example in Figure 2.15 gives details about this operator. Given user selected classes $S = \{C1, C2, C3\}$, while $lca(C1, C2, C3) = \{C1, C2, C3, B1, B2, B3, A1, A2\}$,$lca(C1, C2) = \{B1, B2\}$ and $lca(C1, C3) = \{B2\}$.

U_{lca} operator can not always build a valid hierarchy tree. For example, sample given in Figure 2.16, result of U_{lca} operator is not complete. Ranwez et al. proposes another operator *closure of* U_{lca} for cases like this. This operator computes the closure of the tree and extracts the core of the ontology module. A similar approach is also used for the *gcd* operator and the hierarchy tree is completed.

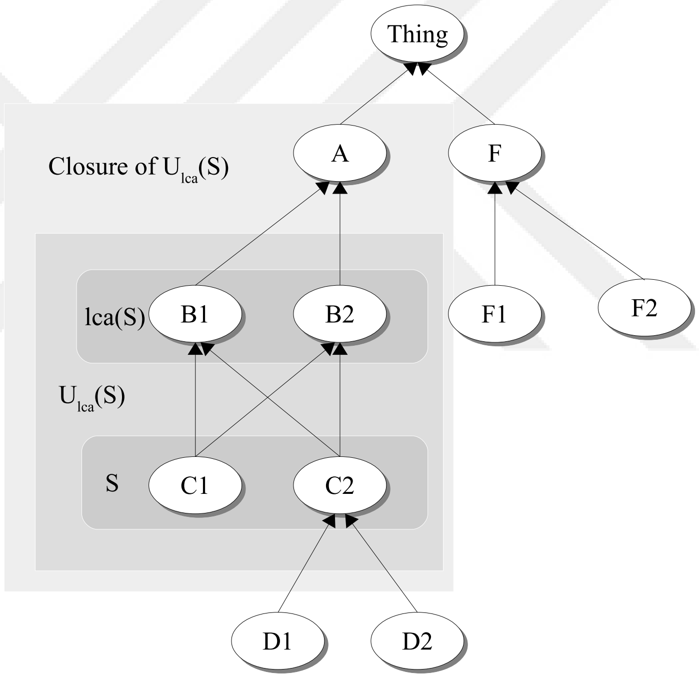


Figure 2.16: An example of closure of U_{lca} (Ranwez et al., 2012).

After extracting the core module using lca and gcd operators, it is possible to add other type of semantic relations according to user needs.

2.5.9 Abis

This method focuses on reducing the problem to sub-graph problem and using semantic relatedness values as distance between concepts (Abiş, 2011). The aim of this study is to overcome the shortcomings of ontology modularization methods extracting ontology modules of minimum size. Within the scope of this thesis work, these shortcomings are listed below:

- All ontology modularization methods are rule based. Therefore, to extract an module that meets user requirements, users need to have in-depth knowledge about domain and have to deal with extraction rules.
- Quality of extracted ontology module depends on how accurately user selects parameters of the algorithm. Most of the time, extracted ontology module goes beyond user needs.
- Ontology modularization algorithms do not take into account the semantic relationships between user-selected concepts.

To solve these problems, a graph-based algorithm has been developed to extract modules from ontologies. This algorithm uses semantic relatedness to find the most semantic paths among selected concepts. Thus, minimal modules can be created that integrate the concepts of user's interest.

Algorithm of module extraction is shown in Algorithm 2.5. As seen in algorithm, *extractOntologyModule* takes two parameters as input: original ontology O and user selected concepts S. First, a randomly selected concept is appended to ontology module M. Then, every possible combination between remaining user selected concepts in S and concepts in M is checked to find the most semantic path. Once the path is found, it's appended to the module and current concept removed from S. These steps continue until there is no item in S.

Ontology module extraction application consists of 3 sub-components. These components are shown in Figure 2.17. Ontology import module is responsible for loading ontologies from external sources into the application. The search module performs the process of searching the concepts in ontology via labels and comments. The module extraction component runs the semantic relatedness-based module extraction algorithm.

This method has been implemented as a desktop application using Jena as ontology processing library and MySql as backend ontology database. All case studies of the method have been performed on SNOMED-CT ontology, but it is also possible to apply it to other ontologies.

Algorithm 2.5 Ontology module extraction algorithm (Abis, 2011).

```
Input: Ontology $O = (V, E)$ which is represented as an RDF graph, user selected
    concepts $S$
Output: Ontology module $M$
function EXTRACTONTOLOGYMODULE(O, S)
    $M \leftarrow$ initialize ontology module as an empty graph
    insert randomly selected item $s_1$ of $S$ into $M$
    remove $s_1$ from $S$

    while $S$ is not empty do
        $s_i \leftarrow$ null
        $path \leftarrow$ null
        $relatedness \leftarrow 0$
        for all concept $m$ in $M$ do
            for all concept $s$ in $S$ do
                $p \leftarrow FindPathBetween(m, s)$
                $r \leftarrow FindRelatednessOfPath(p)$
                if $r > relatedness$ then
                    $s_i \leftarrow s$
                    $relatedness \leftarrow r$
                    $path \leftarrow p$
                end if
            end for
        end for
        $AppendPath(M, p)$
        remove $s_i$ from $S$
    end while
    return $M$
end function
```

It is not possible to use this method and application in doctorate thesis work because of the following problems:

- The application works on Jena and MySql. Therefore, scalability and performance problems will occur under heavy load.
- It's not possible to apply sophisticated graph algorithms on Jena and MySql.
- There are limitations in the traversal algorithm due to possible performance problems.
- Although algorithm is semantic relatedness based, semantically correct path restrictions are not used (see Section 4.3.1.2).

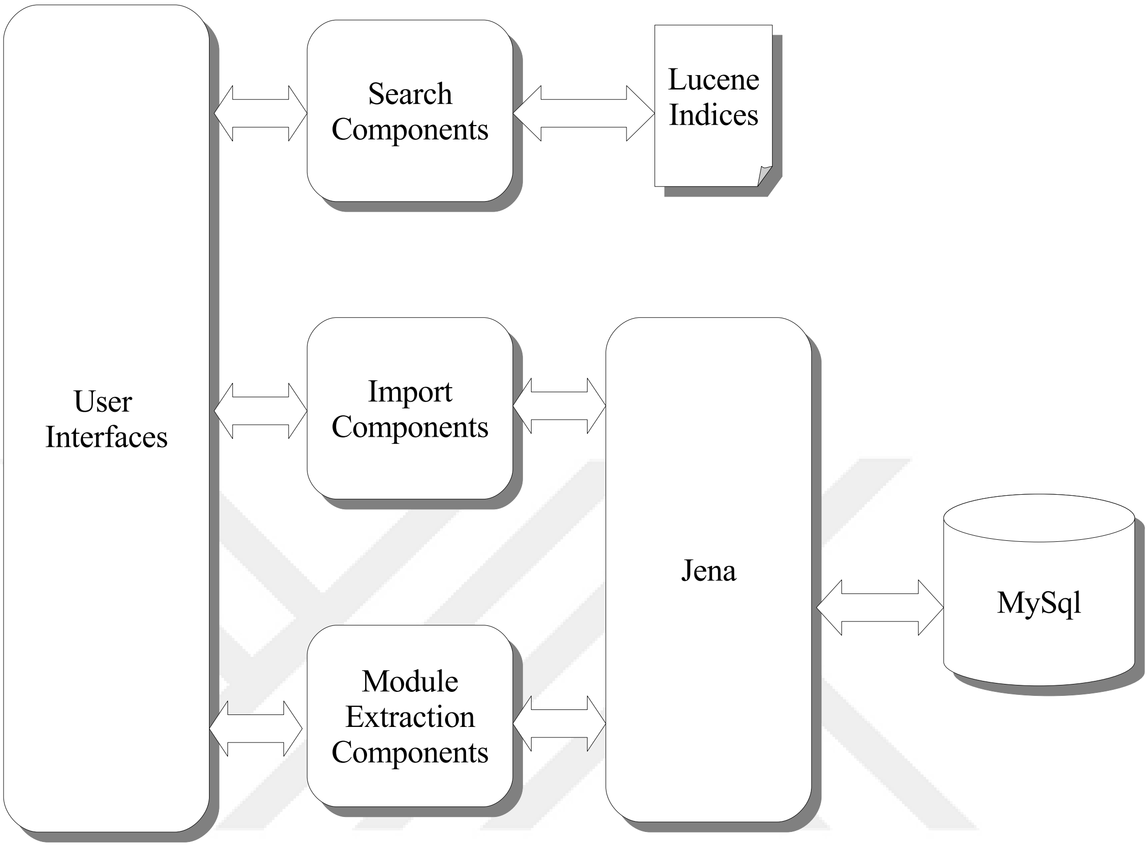


Figure 2.17: Architecture of module extraction application (Abis, 2011)

- It is not suitable for use in an incremental reuse process.
- No API support provided for external use.

2.5.10 Comparison of ontology modularization methods

Ontology modularization plays a key role in the reuse of large ontologies. Because it is not feasible to import an ontology which has hundreds of thousands of concepts into a newly developed ontology, one should consider the modularization techniques.

In the ontology development literature, there are several studies that have proposed a modularization method. As mentioned in the previous sections, these methods generally fall into two groups: partitioning and module extraction. In this thesis study, the module extraction techniques are emphasized, because of their dynamic structure and their ability to adapt more easily to an iterative process.

In this section, all module extraction techniques, which are detailed in this study, are compared according to some criteria that are important for ontology reuse (See Table2.9). Details of these criteria are listed below:

- **Type** specifies how the ontology modularization method works.Many methods are traversal-based and traverse ontology using predefined and user selected rules. However, some methods use graph operators while some methods use SPARQL queries.
- **Concept selection/filtering** indicates how many concept can be selected by user to extract an ontology module. All methods allow users to select multiple concepts, except Doran et al. (2007).
- **Recursive class hierarchy** criteria specifies whether all the super-classes of a concept are recursively included in the module.
- **Semantic completeness** criteria specifies whether semantic completeness rules are executed for each concept to make extracted ontology module semantically complete.
- **Consistency check** criteria specifies whether extracted ontology module is checked for possible inconsistencies.
- **Semantic relatedness** indicates whether semantic relatedness values among concepts are used during module extraction.
- **GUI support** is also important for end-users. Many methods do not offer user interfaces or plugins.
- **API support** is important to support external access and increases usability of method.
- **Iterative** criteria specifies whether ontology modularization method supports iterative reuse processes.
- **Methodology integration** specifies whether ontology module extraction method integrated with a ontology development methodology or ontology reuse process.

Table 2.9: Comparison of ontology modularization methods.

	Type	Concept selection	Property selection	Recursive class hierarchy	Semantic completeness	Consistency check	Semantic Relatedness	GUI support	API support	Iterative	Methodology integration
Noy & Musen	Traversal-based	Multiple	Yes	Yes	Yes	No	No	Yes	No	No	No
Seidenberg & Rector	Traversal-based	Multiple	Yes	Yes	Yes	No	No	No	No	No	No
Doran et al.	Traversal-based	Single	No	Yes	Yes	Yes	No	Yes	No	No	Yes
MOVE	Traversal-based	Multiple	Yes	Yes	Yes	Yes	No	No	No	No	No
Miao et al.	Graph-based	Multiple	No	No	No	No	No	No	No	No	No
SOMET	Query-based	Multiple	Yes	N/A	N/A	N/A	No	No	No	No	N/A
Hussain & Abidi	Traversal-based	Multiple	No	No	Yes	No	No	No	No	No	No
Ranwez at al.	Graph-based	Multiple	No	No	No	No	No	No	No	No	No
Abis	Graph-based	Multiple	No	No	Yes	No	Yes	Yes	No	No	No

Most of these modularization methods work in a traversal approach. They convert ontology into their internal representation and build ontology module using traversals. It is also possible to categorize Miao et al. (2008) and Ranwez et al. (2012) as traversal-based. However, since they heavily use graph theory, it will be clear to put these methods into graph-based category which is a sub-category of traversal-based. On the other hand, SOMET works in a query-based approach. Although it is not a new method for module extraction, it can integrate existing methods via SPARQL queries.

All these methods build ontology modules around user selected concepts. Some of these methods also support property selection and filtering. Completeness and consistency are important topics of ontology modularization methods. Many approaches prefer to adhere to these two criteria. Because applying completeness and consistency check directly increases the module size, it is hard to balance between module size and user preferences. One of the solutions to this problem has been proposed by Hussain and Abidi (2009). Instead of including superclasses recursively, they prefer to import definitions of these super-classes into user selected classes directly.

The concepts chosen by the users mostly refer to the core of the ontology module to be created. The point in the mind of the user, that he wants to reach first, is the sub-space that connects only these concepts. Therefore, in addition to completeness and consistency, it is also very important to consider the closest (most semantic) paths among these concepts. However, these relationships should not be considered only taxonomically (using only superclass and subclass relations), but also all the other types of relations. Once this kernel module has been created, items that will meet criteria such as completeness and consistency should be left to the user. Thus, users can clearly define the borders of their ontology module.

GUI and API support is one of the major shortcomings of existing methods. While only a few methods provide user interfaces, no method has API support. All methods work on their local processes.

While ontology development is an iterative process, ontology reuse should also be performed in an iterative manner. Ontology modularization methods should also be compatible with this approach. For example, the user must be able to add new concepts to an ontology module that he/she has created in the previous iteration. The module extraction process should not be operated from the beginning. However, the existing methods are not suitable for an iterative approach. Also, except Doran et al. (2007), none of these methods has proposed a methodology for ontology reuse.

2.6 Conclusion

In this section, detailed information has been given about ontologies and ontology reuse. In addition, topics which are important for ontology reuse, such as ontology development methodologies, ontology reuse tools and ontology modularization methods, have been examined considering their current status and shortcomings. Finally, once again, this conclusion section will provide information that summarizes the topics covered in this section.

The right thing to say about ontology reuse is that, despite the awareness of all researchers, the desired point has not yet been achieved. This is valid for all sub-parts of ontology reuse process.

It is not possible to say that ontology development methodologies are matured and well-accepted as in software development. However, there are many ontology development methodologies in the literature that have emerged from different perspectives and experiences. These methodologies often include ontology reuse as a step and emphasizes its importance. However, they do not provide any recommendations and guidelines. The difficulties and requirements of ontology reuse make it an iterative and incremental process, and unlike existing methodologies, this process should take place as a sub-process of the ontology development. If possible, it should be supported by the ontology reuse tools which are compatible with the preferred ontology development methodology.

Another important part of ontology reuse is ontology reuse tools. There are not many studies on ontology development tools in the literature. Also, most of these studies are outdated and unusable. Those which are up-to-date generally do not have sufficient functionality. They only allow importing of a single concept and its definition into the new ontology. It is not possible to create an ontology module with more than one concept using these tools. Only Ontofox supports multiple concept selection, but it is not truely an ontology modularization support. Also, none of these tools provide an API to enable integration with other applications. Lastly, they do not have an graph-based traversal user interface to allow users to understand the relations between selected concepts better.

Ontology modularization is also an important topic of ontology reuse, especially when reuse of huge ontologies is required. All of these approaches focus on the extraction of a valid and consistent ontology module. Most semantic paths among selected concepts are not taken into consideration primarily. These extracted ontology modules can go beyond reuse requirements when linked-data are considered. Also, existing ontology modularization methods do not correspond to the iterative and incremental characteristics of the reuse process. Lastly, one of the most important shortcomings is the lack of API and tool support.

One of the most important problems of ontology reuse is the lack of a integrated solution that connects methodologies, methods and tools, which is also the main purpose of this study alongside with other shortcomings (see Section 2.6 and 2.3.3).

3. A REUSE-BASED AND AGILE ONTOLOGY DEVELOPMENT METHODOLOGY

The aim of this thesis study is to define a new ontology development methodology and ontology reuse process which are supported by a ontology modularization method and a reuse tool. Ontology reuse process and modularization method are intended to be semi-automatic as stated by Kamdar et al. (2016) and to eliminate the current shortcomings. Although these shortcomings are detailed in the previous section, it is possible to summarize them as follows:

- Although ontology development methodologies emphasize the importance of ontology reuse, they do not provide any guidance about the details of reuse.
- Ontology reuse tools are not sufficient to meet reuse requirements. Many of the reuse tools that are currently active only support importing a single concept and its definition into new ontology.
- Modularization methods are not capable of supporting ontology reuse as a process. They are not suitable for iterative and incremental approaches.
- Lastly, there is no approach that integrates an appropriate methodology, a reuse process, a modularization method and a reuse tool.

All these shortcomings stand as a barrier to ontology reuse and prevent its spread. However, if properly supported, ontology reuse can accelerate development and reduce costs, as well as improve the quality of ontologies developed.

Ontology reuse is a process with its own sub-steps, such as search, evaluate, import, integrate and modularize. All these steps should be performed in an iterative and incremental fashion. But, existing methodologies do not have a complete a reuse process and do not meet these requirements. Therefore, there is a need for a new and novel ontology development methodology, which acts as a framework for ontology reuse process and ontology modularization method to be developed in the scope of the thesis. This methodology should heavily focus on reuse to make the requirements of ontology reuse and modularization clear.

This section firstly examines the steps of ontology reuse processes in the literature. Then, an iterative and incremental reuse process is defined. Ontology development methodology has been created based on this process and all sub-steps of the methodology are detailed in the following sub-sections.

3.1 Proposal for an Agile Ontology Development Process Supporting Ontology Reuse

As mentioned above, nearly all of the ontology development methodologies in literature emphasize reuse without any detail. Often, methodologies are defined as just guidelines that do not go into too much detail. For example, Ontology Development 101 methodology recommends reuse as second step of development and only talks about the importance of reuse giving some examples.

Ontology reuse should not be defined only as a sub-step of ontology development methodology, but rather as a separate process. Then, it should be integrated with ontology development methodology. Reuse is inherently iterative and incremental. It also directly affects all the other steps and requirements of the ontology development (Abiş et al., 2019). This can be explained by an example. Suppose that, an ontology engineer is developing a new ontology about cities. She/he enlisted important terms about the domain. It is highly possible that one of these terms is *City* itself. Because there are many well-accepted general-purpose ontologies such as Dbpedia, Wikidata and Schema.org, she/he thinks that it's better to reuse terms from these existing ontologies. She/he checked Dbpedia, Wikidata and Schema.org for *City* term and found definitions given in Figure 3.1.

Regardless of which definition is chosen, especially if super-class relationships are also selected, it is highly possible that new concepts will be added to ontology which are not listed in requirement documents such as *Settlement* or *Place*. This will, at least, lead to a review of the requirement documents and affect the next iteration directly.

3.2 An Iterative and Incremental Reuse Process

Although existing ontology development methodologies does not give too much details about ontology reuse, there are some studies that focus on ontology reuse process. These studies are important to understand reuse steps and

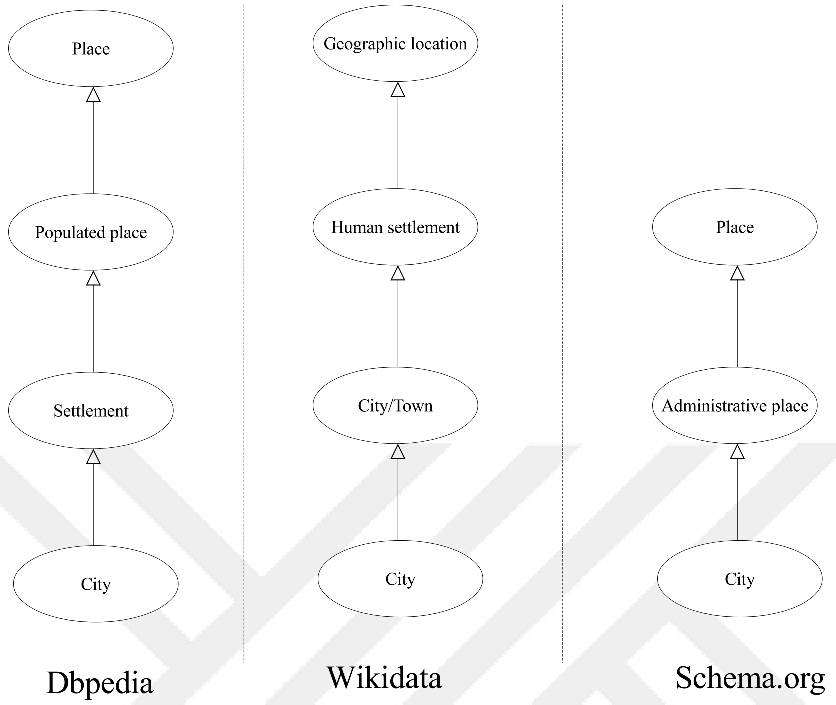


Figure 3.1: City class definition in Dbpedia, Wikidata and Schema.org.

requirements. As detailed in Table 3.1, most of these processes have similar steps.

The reuse steps listed in Table 3.1 have been examined in detail. Common reuse steps suggested by different studies have been identified, and considering these steps, an iterative and incremental ontology reuse process has been defined. The steps of this process are shown in Figure 3.2 and detailed below.

- **Search:** Ontology engineers need to search for possible ontologies to reuse.
- **Select**: Ontology engineers need to evaluate candidate ontologies and choose the most appropriate one.
- **Find:** Ontology engineers need to find possible terms to reuse in selected ontologies.
- **Build:** Selected terms and relations should be extracted from selected ontology using techniques such as path extraction and module extraction.

Table 3.1: Reuse processes in literature.

Research	The steps of reuse process
(Uschold et al., 1998)	1-Understand the ontology to be imported and find relevant parts 2-Ontology translation 3-Refinement 4-Verification 5-Integration
(Pinto and Martins, 2000)	1-Find and choose candidate ontologies 2-Evaluate candidate ontologies 3-Asses candidate ontologies 4-Select the most appropriate ontology for reuse requirements 5-Integrate selected ontology into new ontology 6-Asses resulting ontology
(Russ et al., 1999)	1-Find and select candidate ontologies 2-Translate ontology 3-Merge ontologies 4-Integrate ontology
(Arpírez et al., 2000)	1-Choose candidate ontologies 2-Analysis candidate ontologies 3-Integrate
(Peralta et al., 2003)	1-Choose possible ontologies to reuse 2-Reverse engineering 3-Import 4-Analysis and revision

- **Integrate:** Extracted parts should be integrated into newly developed ontology.
- **Evaluate:** Resulting ontology should be evaluated against requirements before new iteration.

3.3 Agile Ontology Development 101

The reuse process detailed in previous section (see Section 3.1) should be integrated into an iterative and incremental ontology development methodology. For this purpose, an agile ontology development methodology is proposed considering the reuse steps and the success of agile software development methodologies. This methodology is compatible with reuse as well as the ontology development principles of previous methodologies. The steps of this methodology is given in Figure 3.3.

It is not practicable to define an ontology development methodology only considering the reuse process and its steps. The relationship among reuse process and the other steps of the methodology should also be defined clearly. Therefore,

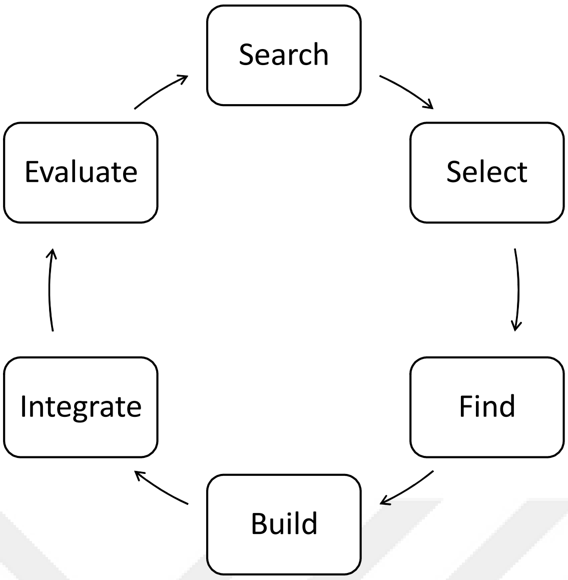


Figure 3.2: New reuse process.

a new and complete methodology has been defined taking into account the existing ontology development methodologies in the literature along with the reuse steps.

With the Agile Manifesto[23] published in 2001, agile software development has started to gain importance, and many methodologies such as Scrum[24] and Extreme Programming (XP)[25] have become widespread (Abiş et al., 2019). The general principle of agile processes is to keep up with the change at every stage of development. This is achieved through an iterative and incremental life cycle. The ability to keep up with change is also valid and critical for ontologies. The knowledge is constantly updated and the use of existing ontological resources changes current requirements. Therefore, adapting the success of agile software development methodologies to ontology development is an important contribution. Existing studies, such as (Peroni, 2017) and (Keet and Ławrynowicz, 2016), have already presented different recommendations and methods for developing an agile ontology development methodology. Unlike other studies, Agile Ontology Development 101 includes reuse as a detailed process with an agile perspective.

The Agile Ontology Development 101 methodology consists of four sub-processes, one of which is the main iteration: (1) Scope and requirement analysis, (2) development, (3) reuse and (4) modular reuse (see Figure 3.3). The

[23] https://agilemanifesto.org/
[24] https://www.scrum.org/
[25] http://www.extremeprogramming.org/

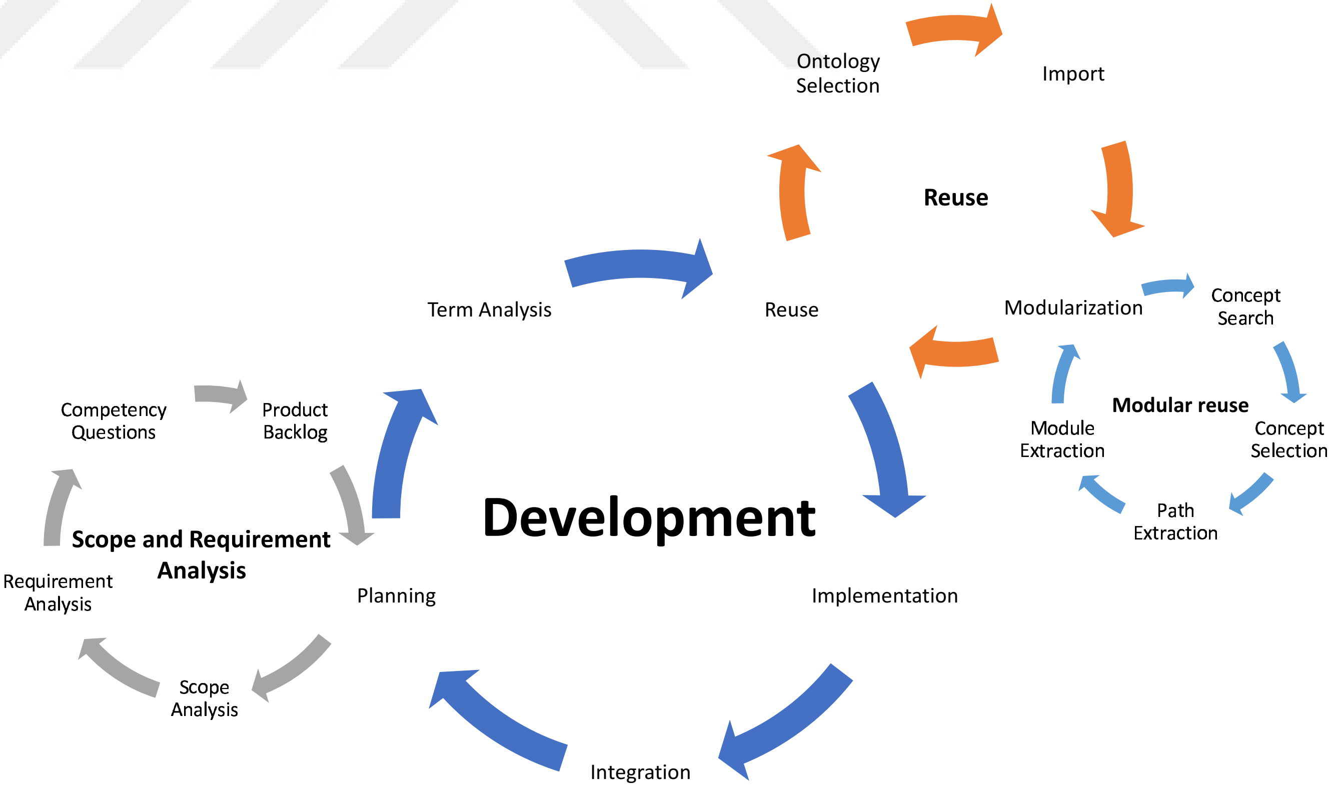


Figure 3.3: Agile ontology development 101.

steps in each child iteration can be repeated multiple times, as in the main iteration. Once the corresponding iteration is complete, the flow can continue in the same cycle or return to the next step of the parent iteration.

Agile Ontology Development 101 is a framework rather than a detailed guideline. Project management details are not included with it. Each activity is a recommendation for ontology engineers. How to implement the activities is left to ontology engineers and project managers. This approach is also consistent with agile methodologies. Another point to be emphasized is that validation and application of this agile ontology methodology is beyond the scope of this thesis. This effort has been made to better identify and validate the needs of the reuse process and modularization method.

3.3.1 Scope and Requirement Analysis

Scope and requirement analysis is positioned as a sub-iteration that corresponds to the detail of the planning activity of the development iteration. Scope and requirement analysis is the first step of ontology development. However, against possible updates, it can be repeated as the first step of each development iteration.

Scope and requirement analysis includes the activities of the first step (Determine the domain and scope of the ontology) of Ontology Development 101 (Noy and McGuinness, 2001). It corresponds to activities within planning and scope analysis in other methodologies. The sub-steps of this iteration are as follows:

3.3.1.1 Scope analysis

Firstly, the scope and domain of the ontology should be determined. It may be useful to answer some predefined questions to get more details about the ontology. Noy and McGuinness (2001) has proposed following examples for these types of questions:

- What is the coverage of ontology domain?
- For what purpose ontology will be used?
- What type of questions should be answered by the ontology?

- Who is going to use ontology?
- Who is going to maintain ontology?

3.3.1.2 Requirement analysis

After completion of the scope analysis, a more detailed study about the content and requirements of ontology should be carried out. Use cases (Kendall and McGuinness, 2019) or motivation scenarios (Grüninger and Fox, 1995) may be preferred to elaborate these requirements.

3.3.1.3 Competency questions

Competency questions express the questions that an ontology should answer (Noy and McGuinness, 2001). In other words, they are informal specification of formal axioms that ontology will cover. Therefore, defining these questions is important in terms of requirements and validation of ontology. Competency questions can be obtained from scenarios identified in the previous stage. Competency questions can be defined in different ways, depending on preference. However, it should include at least an informal question and the answer to that question. An example competency question is shown below (Peroni, 2017):

Table 3.2: An example competency question (Peroni, 2017).

Identifier	3
Question	What are the types of vehicles?
Answer	All vehicles used in land, vehicle, sea, air and rail transport
Examples	Car, bus, train, ship, ferryboat, plane, helicopter
Related Questions	1, 2

3.3.1.4 Building product backlog

In agile software development methodologies, product backlog items and tasks are stored in a pool called as Product Backlog. Similarly, the pool where the defined scenarios and competency questions will be stored is also called as Product Backlog in this methodology. Keeping this backlog up-to-date is very crucial for the progress of the ontology development process. Therefore, the backlog items should be reviewed and verified in each scope and requirement analysis iteration.

3.3.2 Development

Development phase is the main sub-process of all ontology development life cycle. All other sub-processes are performed as a part of development. At the end of every development iteration, a valid ontology part should be released. Validation of this release is checked with the competency questions, that are defined in planning (scope and requirement analysis) phase and selected for this iteration. All the steps of the Ontology Development 101 guideline, except the domain and scope analysis, are carried out in this sub-process.

The development stage does not give details such as how to develop ontology, what tools should be used, and which formal language ontology should be preferred. The phases of the development process are presented in a more abstract point of view with a focus on reuse.

The development phase is organized into 5 sub-steps:

3.3.2.1 Planning

The first step of development is iteration planning. Planing is a complex step and corresponds to scope and requirement analysis process. Initially, the scope and requirements analysis process should be executed to specify requirements and put them into the product backlog. While planning step is a process of ontology development methodology, it can be repeated in an iterative fashion, until all requirements of current development iteration is met. Planning is not only limited to initial development iteration, but also all iterations to review and refactor requirements. End the end of planning, competency questions which are requirements of current development iteration, are selected. Selected competency questions should be registered into development backlog.

3.3.2.2 Term analysis

Within the scope of selected competency questions, the terms to be modeled and implemented within the current iteration should be determined. This step is similar to the "enumerate important terms" step in Ontology development 101 (Noy and McGuinness, 2001).

3.3.2.3 Reuse

Candidate ontologies to be reused can be partially determined during the scope and requirement analysis process. However, competency questions and enumerated terms are more determinant to find existing ontologies in similar domains. Ontology reuse is often not elaborated in other methodologies and is involved in planning stages. For example, Ontology Development 101 defines reuse just before development, as the second step of all methodology. Agile Ontology Development 101, which is defined in this thesis, details reuse and defines it as a process that can be iterated multiple times.

3.3.2.4 Implementation

As a result of this step, an ontology which is formalized in a ontology language such as OWL or RDF/S should be implemented. All taxonomic relations and other axioms should be defined in this step. Ontology engineers can also carry out other activities like conceptualization. Agile Ontology Development 101 does not elaborate the implementation step too much and allow ontology engineers to choose detailed activities. The implementation step is much more detailed in other methodologies and usually divided into sub-steps. For example; all the steps after the second step (reuse step) of Ontology Development 101 correspond to implementation (Noy and McGuinness, 2001).

3.3.2.5 Integration

The ontology part, which is output of implementation step, should be integrated with the main ontology, that is obtained and integrated in previous iterations. After that, main ontology should be validated against the all competency questions which are already in development backlog.

3.3.3 Ontology reuse

Ontology reuse is the main focus of Agile Ontology Development 101 methodology and is a process that affects and changes the entire ontology development. Therefore, it should be performed as a part of the main development process and carried out in an iterative way. Ontology reuse should also be supported with semi-automatic modularization methods and reuse tools.

Nearly all of the existing ontology development methodologies do not give details about reuse. For example; Ontology Development 101 includes reuse as the second step of all process (Noy and McGuinness, 2001). But, Agile Ontology Development 101, which is defined in this thesis, elaborates reuse process and has detailed sub-steps.

Ontology reuse process of Agile Ontology Development 101 can be repeated as subsequent iterations until all the requirements of current development iteration met. These requirements are dependent to ontology engineers and Product Backlog which is defined in the scope and requirements analysis process. There are no strictly defined rules in this methodology to evaluate reuse requirements.

Sub-steps of ontology reuse process are detailed below:

3.3.3.1 Ontology selection

First step of reuse is finding existing ontologies. Scope and requirement analysis, competency questions and term lists give clues to find right ontological resources. It is possible to find multiple candidate ontologies while searching. Techniques like ontology evaluation can be used to evaluate and filter automatically found results (Jonquet et al., 2010) (Martínez-Romero et al., 2014) (Martínez-Romero et al., 2017).

3.3.3.2 Import

Simplest form of reuse is importing existing ontologies. Nearly all formal languages and ontology development tools give constructs to import ontologies. Import is usable for small and medium sized ontologies, especially for upper-level ontologies like BFO and Dublin-core. But this method is not useful for large-scale ontologies.

3.3.3.3 Modularization

Large-scale ontologies can not be imported directly because of problems like performance, visualization and reasoning. In these cases, modularization techniques come into play. Agile Ontology Development 101 defines modularization due to its complexity.

3.3.4 Ontology reuse by modularization

As stated in previous section (Section 3.3.3), ontology modularization is sub-process of ontology reuse process, and it must be supported with a semi-automatic ontology modularization method and reuse tool. Ontology modularization should also be an iterative sub-process. Ontology engineers can extract an initial ontology module in the first iteration, and can extent this module repeating module extraction in subsequent iterations by adding new concepts or paths.

Ontology modularization sub-process can not be applied by ontology engineers without any method and tool support. Therefore, this sub-process must be supported with a novel and iterative ontology modularization method. Details of this modularization method and reuse tool are given in Section 4, 5 and 6.

Modularization sub-process is actually a tool dependent step, and it gives clues about requirements of ontology modularization method. Modularization sub-process contains three sub-steps:

3.3.4.1 Concept search

Interested terms should be searched in selected ontologies by user. This step depends on the capabilities of the ontology reuse tool to be used. It can be carried out more easily if the relevant tool provides full-text search functionality.

3.3.4.2 Concept selection

Relevant concepts should be selected by user. The selected concepts will form the starting point for the ontology module. Concept selection is not just a choice of a single concept. Previously extracted paths and ontology modules should also be included in the selection. Thus, it can be ensured that the ontology modularization sub-process is iterative.

3.3.4.3 Path and ontology module extraction

Modularization methods should be operated on selected concepts to form the path or ontology module. At this step, new and iterative modularization methods are needed to support reuse process.

3.3.5 Conclusion

The reuse-based and agile ontology development methodology defined above reveals the basic needs of ontology reuse process and modularization method. It is possible to summarize these needs as follows:

- Reuse of ontology can affect the entire development. It should therefore be included in each development iteration. Also, reuse should be operated as an iterative and incremental process.
- It should be possible to search within ontologies and compare the search results.
- Upper-level ontologies should be be imported directly.

- It should be able to extract a single concept with its definition from large-scale ontologies.
- It should be able to extract ontology modules from large-scale ontologies.
- The method and tool to support ontology reuse should be semi-automated to allow the intervention of ontology engineer.

4. DEVELOPING A MODULARIZATION METHOD TO SUPPORT ONTOLOGY REUSE

The scope of reuse planned for this thesis study is shown in Figure 4.1. Apart from all these, also there are many requirements such as ontology evaluation, ontology mapping and ontology alignment to support ontology reuse, but all these studies are not included in the scope of the thesis. Ontology development tools often provide constructs to import existing ontologies. Therefore, there is no need to carry out a study within the scope of this need.

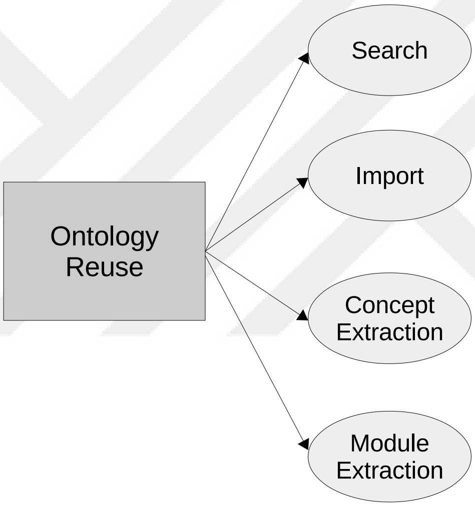


Figure 4.1: The scope of reuse.

Searching for concepts in existing ontologies is an architectural problem rather than a methodological approach. Therefore, the solution to the search problem is explained in Section 5.

The aim of the study to be proposed in this section is to reveal a ontology modularization method based on semantic relatedness. First of all, the need for such a method should be explained.

4.1 The Need for Yet Another Ontology Modularization Approach

Before specifying why the available methods are not sufficient, we should emphasize that there is no accepted definition about "what a good ontology module is?" (d'Aquin et al., 2009) (Doran et al., 2007). Therefore, all of the existing methods propose their own approaches using subjective assumptions (Doran et al., 2007). The only one who can judge whether the extracted module is sufficient or not, is the users themselves. So, the ontology modularization method should involve the user in modularization using a semi-automatic approach, and should try to extract a minimum starting point for initial ontology that connects all user selected concepts in the most semantically way. Then, user can expand or collapse this module according to their requirements.

First shortcoming of existing methods is that, they are not purely reuse oriented (see Section 8.2 for all shortcomings of modularization methods). Instead, they propose general purpose modularization approaches, which are impractical to integrate with an iterative and semi-automatic reuse process. Once the ontology module has been extracted, user should either continue with this module or start from initial by changing all the parameters. It is not possible to extent currently extracted module in subsequent iterations and connect with previously extracted modules and paths.

Another shortcoming of existing methods is about how they built the ontology module and ontology module size. All modularization methods heavily focus on pre-defined rules and semantic completeness rather than relations between selected concepts. As a result, the extracted module includes more than user requirements and complicates reuse process. According to MIREOT principles (Courtot et al., 2011), the minimum amount of information to reference a term from an external ontology is only URI of ontology and URI of term. Nevertheless, concerns about semantic completeness make ontology modules bigger. For example; Bhatt et al. (2006) includes all super-classes of selected concepts using rules called Semantic Completeness Optimization Schemes (SCOS). Also, approaches proposed by Noy & Musen (2004) and Seidenberg & Rector (2006) considers super-classes recursively.

To clarify the minimum information approach and importance of relations among selected concepts, let us give an example. Suppose that, an ontology engineer is developing an ontology about vehicles, and competency questions defined in requirement analysis have already provided some terms, such as *car*, *ship*, *train*, *bus* and *aeroplane.* The simplest way to reuse these terms from external sources is referencing them one by one only using their URIs (While it is impracticable to import the whole WordNet ontology, this alternative is automatically ignored). However, engineer has chosen more than one concept together and most likely, he or she will want to observe relations between these concepts before reuse. So, it is important to extract a minimal module that connects only selected concepts together first, as shown in Figure 4.2. Then, they could expand this module using actions such as “add all siblings of car”, “add all super-classes recursively from transport to root class” or “add equivalent classes of all selected classes” (For the sake of simplicity, the figure only includes taxonomic relations and some intermediate nodes are omitted. However, the ontology modularization approach must be able to consider all types of relations).

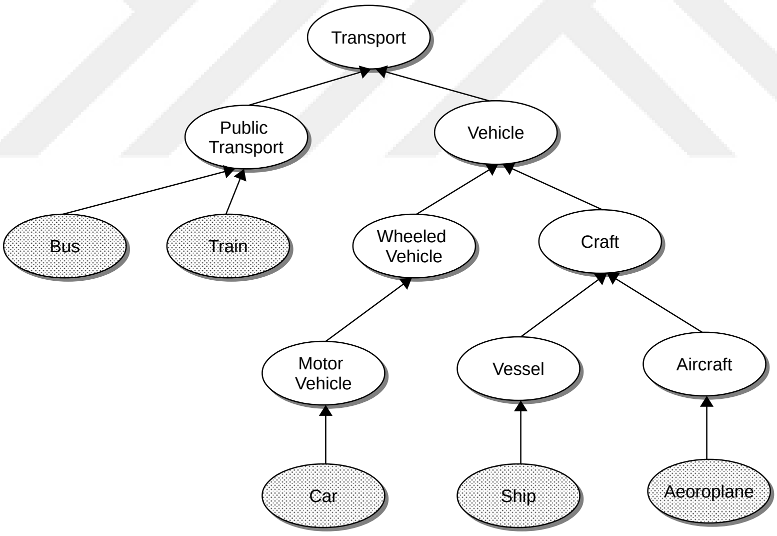


Figure 4.2: A reuse example from WordNet ontology.

Performance, which is the first factor that affects usability, is another problem of existing methods. However, other than Bhatt et al. (2006), there is no study that considers performance alongside with module extraction. Especially, the methods, which uses SPARQL queries directly to expand ontology module graphs, have significant performance problems because of round-trips to databases (These methods execute SPARQL queries to identify possible neighbor concepts, and each SPARQL query brings computation and network overhead). So, designing and developing a scalable and performance oriented method using semantic cloud possibilities is also important for ontology reuse.

Existing ontology modularization methods do not own their ontology registries or are not integrated with existing ontology repositories to enable search from these sources. Users should search and select ontologies, and then they should give selected ontology and other parameters as input to modularization application. Also, it is not practical to extract ontology module from integrated multiple ontologies.

All the existing methods work as standalone local applications. It is not possible to access these applications using APIs like SOAP Web Services or RESTful services. This makes it impossible to integrate these methods with existing applications.

All of the above-mentioned shortcomings raise the need for a new modularization method to support ontology reuse.

4.2 How to build a minimal ontology module?

Many of the current studies handle ontologies as directed graphs and traverse according to predefined rules and user preferences. Some other studies, such as Ranwez et al. (2012), directly benefit from graph operators, such as least common ancestor (lca) and greatest common descendants (gcd). Therefore, handling the ontology modularization as a graph modularization problem and using existing graph algorithms can also facilitate our extraction process.

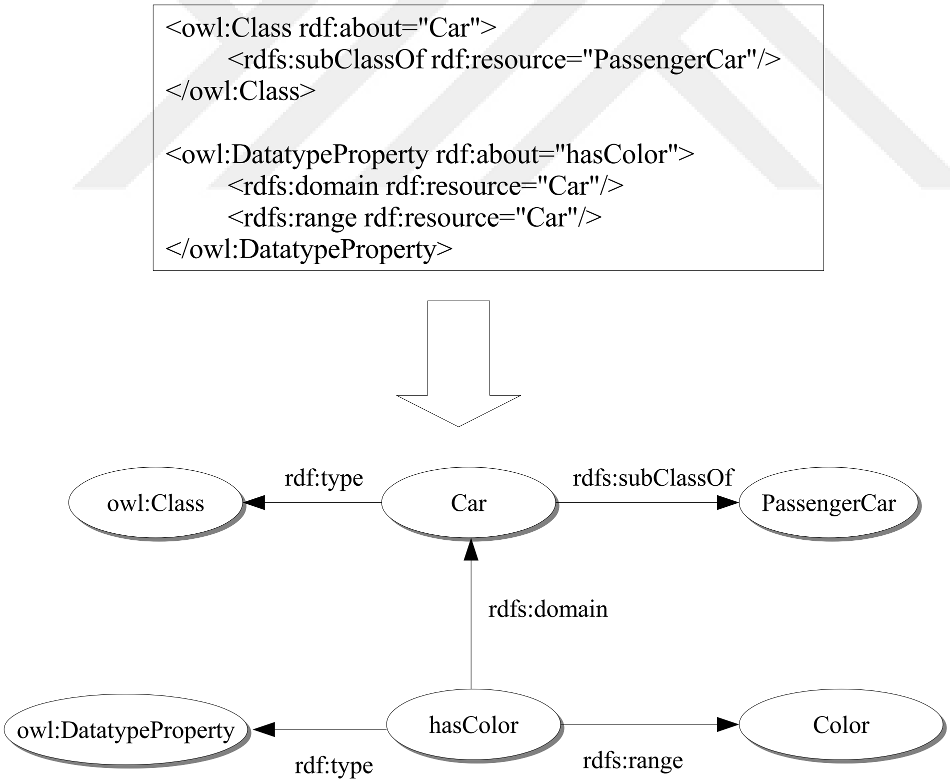
Main purpose of this modularization algorithm is connecting all selected concepts using graph algorithms. Because, it is easier to describe minimal module for general purpose graphs. For example, if two vertices given as v_1 and v_2, the minimum module for these two vertices can be defined as shortest path between them. If more than two vertices selected, module extraction turns into Steiner Tree problem for graphs. Steiner tree problem can be formalized as follows. Given a weighted graph $G = (V, E)$ and a set of selected nodes $V' \subseteq V$, find minimum cost module that contains all vertices of V'.

How the ontology modularization problem can be transformed into a graph problem is explained in the next sections. Firstly, it should be noted that ontologies are actually RDF graphs. Next, the semantic relatedness and how it can be applied to the graphs should be discussed.

4.2.1 Representing ontologies as RDF graphs

It is possible to represent all ontologies as RDF graphs (Hanna et al., 2012). However, these RDF graphs include semantics, unlike general-purpose graphs. Ontologies are also referred as semantic graphs and attributed relational graphs in the literature (Barthelemy et al., 2005) (Coffman et al., 2004). Each node and edge in semantic graphs have a type, and therefore a meaning (Barthelemy et al., 2005).

In order to explain the ontology-graph relationship more clearly, a simple ontology part, with its graph representation is given in Figure 4.3.



```
<owl:Class rdf:about="Car">
        <rdfs:subClassOf rdf:resource="PassengerCar"/>
</owl:Class>

<owl:DatatypeProperty rdf:about="hasColor">
        <rdfs:domain rdf:resource="Car"/>
        <rdfs:range rdf:resource="Car"/>
</owl:DatatypeProperty>
```

Figure 4.3: Graph representation of a sample ontology part.

As shown in the Figure 4.3, each OWL component assigns semantics to the entity that it defines, and these semantics brings additional definitions to RDF graph. For example, *Car* is defined as as *owl:Class* in Figure 4.3. This definition is reflected as a triple between *Car* and *owl:Class* using *rdf:type* relation.

4.2.2 Ontologies and semantic relatedness

Graphs which have weights on each edge are called as weighted graphs. In non-semantic graphs, it is relatively easy to assess these weights. For example; the weight of each edge in a graph which defines the cities will likely be the distance between neighboring cities. However, there is no generalized and well-accepted approach to define the weight between concepts in ontologies. There are studies in the fields of relatedness and similarity in order to achieve such weighting in ontologies and to find the semantic values of paths between concepts.

Relatedness and similarity, which are directly related to each other, are two important topics of artificial intelligence and psychology (Giray and Ünalır, 2013). Similarity is a fundamental component for behavior and knowledge theories and plays an important role in the classification of concepts and objects (Tversky, 1977). Similarity only works on hierarchical relations. On the other hand, relatedness means having a relation or being related (Giray and Ünalır, 2013). Relatedness takes all kind of relations into account between two concepts, such as functional association, meronymy and antonymy (Morris and Hirst, 2004).

Resnik (1995) defines similarity as special form of relatedness and gives *car-gasoline* and *car-bicycle* relationships as an example to clarify these distinction (See Figure 4.4). While car and gasoline are not similar concepts, they are more related than car and bicycle.

Similar concepts are related at the same time. But, the opposite does not always work for relatedness. It is not possible to say every related concept pair is also similar. So, using relatedness in ontologies is a better approach than using similarity, because ontologies have more detailed constructs, such as inverse of, intersection, union and equivalence, rather than simple taxonomies.

There are many studies on semantic relatedness. It is important to give details about these approaches. Then the preferred relatedness method for the ontology modularization method defined in this thesis will be introduced along with reasons.

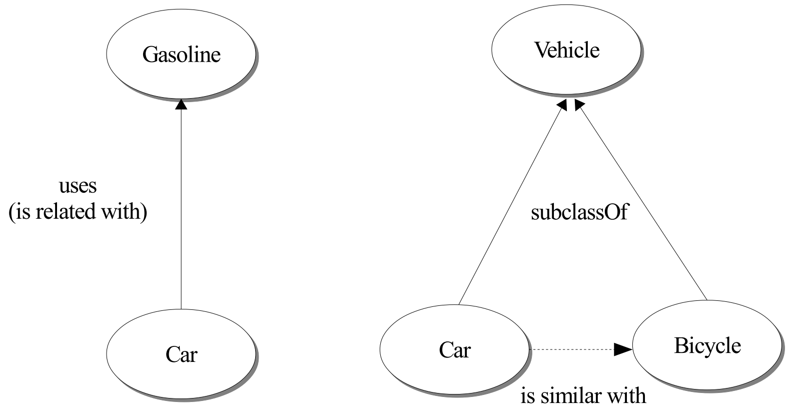


Figure 4.4: Car-gasoline and car-bicycle relations (Resnik, 1995).

While purpose of this thesis is not defining a new relatedness method, existing relatedness methods will be briefly described.

It is possible to divide semantic relatedness and similarity measure methods into five categories: Path-based approaches, ontology-based approaches, information content based approaches, feature-based approaches, and hybrid approaches.

4.2.2.1 Path-based approaches

Path-based approaches use paths in a taxonomy or ontology to calculate the similarity/relatedness measure. Rada et al. (1989) stated that, the similarity between two concepts is directly proportional to the number of edges separating these concepts. The smaller the path between concepts the more similar they are. The Rada approach only uses generalization and specialization links between concepts, so it is also possible to refer this method as *depth-based* (Pirró, 2009).

The simple edge counting approach proposed by Rada et al. has been improved by several studies. Wu and Palmer (1994) have emphasized that similarity should be lower in the upper parts of taxonomy, because that concepts are more general than lower parts. Therefore, they also take into account the depth of the least common ancestor which connects the given concept pair, together with the edge count between these concepts. If depth of sub-taxonomy that connects given concepts is higher, these concepts will be more similar.

Leacock and Chodorow (1998) have proposed another approach based on Rada's study. But, they stated that similarity should increase or decrease in a non-linear fashion and suggested another calculation method. Another path-based approach, which focuses on linked data, has been proposed by Hulpuş and his colleagues (Hulpuş et al., 2015). Their method, which is called exclusivity-based relatedness, heavily works on instances. The rational behind this method is that, a relation is stronger, if each of the concepts is connected fewer than other concepts through the same relation type.

4.2.2.2 Ontology-based approaches

Hirst and St-Orge (1998) has used edge count to implement their similarity method. They do not only focus on taxonomic relations, but also other types of WordNet relations. Hirst and St-Orge has stated that traversing ontologies in all directions without any restriction can cause the loss of semantics. Therefore, they have also suggested restrictions for semantically correct paths.

Giray and Unalir (2012) have proposed a method to measure semantic relatedness, which considers nearly all kind of OWL constructs. Relatedness between a concept pair is calculated by finding all the paths between these concepts. The relatedness value of each path is calculated, and the value of the most related path gives the relatedness value of that concept pair. The relatedness value of a path is calculated by multiplying relatedness values of OWL constructs in that path. The paths can be found with a graph search algorithm such as depth-first search (DFS) or breadth-first search (BFS).

4.2.2.3 Information content based approaches

Due to the limitations of path-based edge counting methods, Resnik has proposed an approach which combines taxonomic relations with the frequency of concepts according to given corpus (Resnik, 1995). Resnik stated that similarity of two concepts depends on the shared information content of these concepts. The shared information content is expressed by the least common subsumer (LCS) of the two concepts. Resnik also states that the information content of a concept can be calculated as the probability of this concept in a text (see Formula 4.1).

$$Probability(C) = \frac{Frequency(C)}{N} \tag{4.1}$$

In Formula 4.1, N is the total number of words in the text, and $Frequency(C)$ indicates the number of appearance of C and its all sub-concepts in the text. Based on the probability function, the information content is calculated as expressed in Formula 4.2.

$$IC(C) = -log\,Probability(C) \tag{4.2}$$

Similarity of two concepts is calculated using IC function. This formula is expressed in Formula 4.3. S states the shared information content of C_1 and C_2.

$$sim_{Resnik}(C_1, C_2) = max_{C \in S(C_1,C_2)} IC(C) \tag{4.3}$$

Lin has improved the approach proposed by Resnik by adding the information content of each concepts as seen in Formula 4.4(Lin, 1998).

$$sim_{Lin} = \frac{2 * sim_{Resnik}(C_1, C_2)}{IC(C_1) + IC(C_2)} \tag{4.4}$$

4.2.2.4 Feature-based approaches

In feature-based methods, unlike path-based and information content based approaches, focus is on the features of concepts. These methods calculates similarity according to shared features of given concepts. Tversky (1977), defines similarity of two concepts $c_1$and $c_2$using three variables:

- Intersection between features of c_1 and c_2
- Set of features when obtained eliminating features of $c_2$from features of c_1
- Set of features when obtained eliminating features of $c_1$from features of c_2

Rodriguez and Egenhofer (2003) have proposed a feature-based similarity measure to calculate similarity between different ontology concepts. This method is based on weighted sum of synsets, features and semantic neighborhood similarities. Sanchez and his colleagues has also proposed a feature-based approach which is based on Tversky's similarity calculcations (Sánchez and Batet, 2013). However, in the calculation of similarity, based on the assumption that ontologies heavily includes taxonomic relations rather than other type of relations, they preferred only sub-class/super-class links over all relations.

4.2.2.5 Hybrid approaches

Hybrid approaches combines the characteristics of different type of information to calculate semantic similarity (Pirró, 2009). One of the hybrid approaches has been proposed by Jiang and Conrath (1997). This approach combines taxonomic edge-counting approach with statistical information which is obtained using a corpus.

Mazuel and Sabouret (2008) has proposed a hybrid approach which based on Hirst and St-Orge's semantically correct path assumption. They uses these restrictions when traversing paths and divides the resulting paths into two groups: single-relation path and mixed-relation path. Single-relation paths only include one type of relation. Mixed-relation paths can include multiple types of relations. They use different similarity calculations for hierarchical and non-hierarchical single relation paths. Similarity of mixed paths are calculated breaking down these paths into single-relation paths.

4.2.3 Comparison of relatedness/similarity methods

The methods detailed above are compared on Table 4.1. Comparison has been made according to the type of method, the relationships considered in traversal and the restrictions used in traversal.

Although many studies have been introduced on semantic relatedness and similarity in literature, the requirements of ontology development method which is defined in this thesis is main criteria to select a relatedness method. These requirements are listed below:

Table 4.1: Comparison of semantic relatedness/similarity approaches.

Method	**Type**	**Type of considered relations**	**Path restriction**
Rada et al. (1989)	Path-based	Taxonomy	No
Wu and Palmer (1994)	Path-based	Taxonomy	No
Leacock and Chodorow(1998)	Path-based	Taxonomy	No
Hulpuş et al. (2015)	Path-based	All types of relations	No
Hirst and St-Orge (1998)	Ontology-based	All WordNet relations	Yes
Giray and Unalir (2012)	Ontology-based	All types of relations	No
Resnik (1995)	Information content based	Taxonomy	No
Lin (1995)	Information content based	Taxonomy	No
Tversky (1977)	Feature-based	All types of relations	No
Rodriguez and Egenhofer	Feature-based	All types of relations	No
Sanchez et al. (2012)	Feature-based	Taxonomy	No
Jiang and Conrath (1998)	Hybrid	Taxonomy	No
Mazuel and Sabouret (2008)	Hybrid	All types of relations	Yes

- The developed method will work on all parts of ontology, including Tbox and Abox. However, the first development will be carried out on Tbox. Therefore, terminological relationships must be supported.

- Since the developed method will be based on path traversal and Steiner Tree, it should work with high performance. It is not possible to make relatedness comparisons using corpus like text. Therefore, information content based methods are out of scope.

- The largest number of possible OWL constructs should be supported.

- Restrictions should be taken into account in path traversals.

Method proposed by Giray and Ünalır (2012) meets many requirements with their semantic relatedness coefficients. However, this method does not take into account the path restrictions in calculations. These kind of restrictions has been used by Hirst and St-Orge (1998). In this thesis, the OWL relatedness coefficients proposed by Giray and Ünalır will be used in integration with the path restrictions of Hirst and St-Orge.

4.3 Semantic relatedness based ontology modularization method

The ontology modularization method proposed within the scope of this thesis is based on the representation of ontology as an RDF graph and the application of the shortest-path and Steiner Tree problems. Thus, it is planned to define a minimal ontology module extraction method using semantic relatedness to facilitate the reuse of existing ontologies. The abstract architecture obtained by the graph algorithms mentioned in the previous sections is illustrated in Figure 4.5.

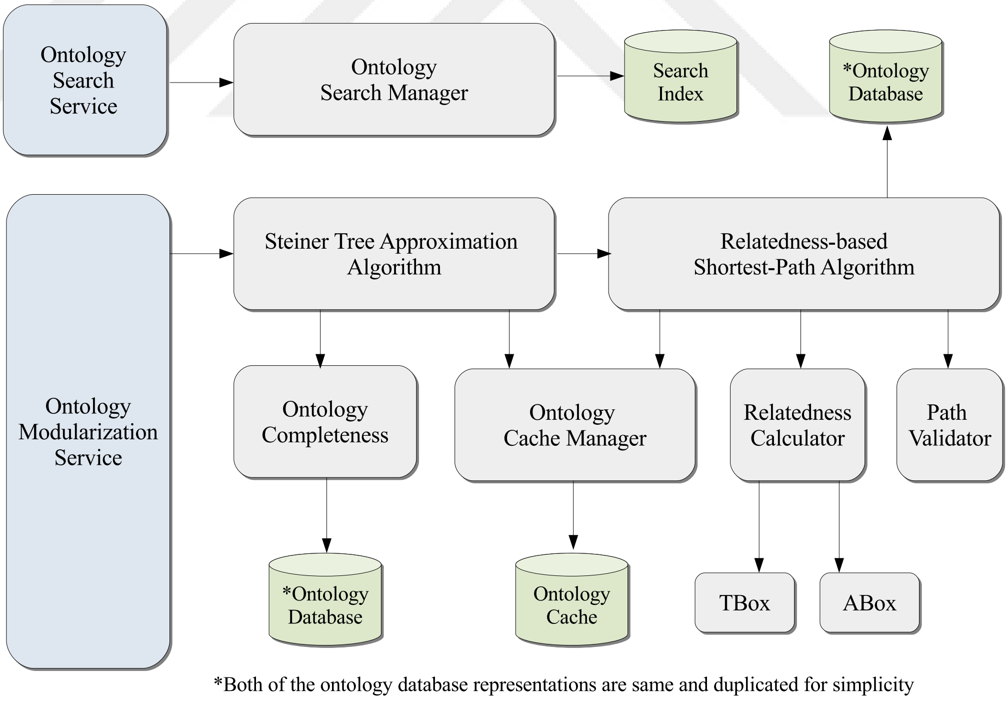


Figure 4.5: Abstract architecture of ontology modularization method.

As mentioned in section 2.5.9, our focus on master thesis was proposing a relatedness-based ontology module extraction method. However, there are some shortcomings due to the scope and possibilities of the study at that time and it cannot be adapted directly to our new method. These shortcomings are listed again below to better understand the details of that study:

- MySql has been used as data storage technology and Jena has been used as ontology processing library. Cloud architectures are not supported and it is not possible to scale it under heavy-load.
- No path restrictions in the literature has been used. Only upward and horizontal relationships are used to make path traversal.
- It is not possible to use it in an iterative process.
- No API support for external applications.

Due to these deficiencies, it is not possible to improve master thesis study. Based on the principles of this study, a new and novel method should be developed.

The newly proposed method consists of two sub-sections: search and modularization. Although search services are an important part of the proposed method, they are more concerned with the architectural and implementation phases. Therefore, only the details of the modularization part will be explained in this section, and it will be assumed that the concepts provided as input to the algorithm have already been found using the search service. Search service and its components will be detailed in the Section 5, where design and implementation of semantic cloud architecture will be discussed.

The most important part of the ontology modularization method is the adaptation of Steiner Tree problem to ontologies, which is used to extract minimal modules from graphs. For Steiner tree problem, the shortest-path approximation method proposed by Takahashi and Matsuyama is used to solve Steiner Tree problem on ontologies (Takahashi and Matsuyama, 1980). While Takahashi and Matsuyama's method is based on shortest-path approximation, it is also important to adapt shortest-path problem to ontologies to find most meaningful path between given concept pairs. This requirement is solved by the application of semantic relatedness.

In order to provide additional performance improvements , caching is also implemented nearly for all stages and layers of the modularization method.

Although the modularization process starts with executing Steiner Tree algorithm and later this algorithm uses shortest path solution, the section will be organized in accordance with the bottom-up approach and firstly, the shortest path problem will be discussed.

4.3.1 Path finding algorithm

The semantic path finding algorithm uses BFS to find paths among two concepts. In order to perform search more efficiently, BFS is applied bidirectionally. It is also necessary to weight each edge on the RDF graph, and as stated in the previous sections, this weighting can be adapted using semantic relatedness values. For this purpose, OWL semantic relatedness values proposed by Giray and Unalir (2012) are preferred.

Although the primary requirement of the path algorithm is to find the most meaningful path, this process should be carried out considering the performance criteria. Otherwise, it is highly possible that usability of the method will become a problem. For this purpose, caching is performed for the paths found and it is aimed to speed up the subsequent searches using cached content.

The pseudo-code of the algorithm is given in Algorithm 4.1 and Algorithm 4.2. Details of these algorithms will be given in the following sections.

Algorithm 4.1 Finding most semantic path algorithm.

```
function FINDPATH($c_1$, $c_2$, $L$, $K$)
Input: $v_1$ is the start concept of the path, $v_2$ is the end concept of the path, $L$ is the maximum depth to search, $K$ is maximum number of paths to calculate relatedness (top-k shortest path)
Output: $P$, if found, most semantic path between $v_1$ and $v_2$, otherwise null
    $P \leftarrow$ empty path object
    $P_{all} \leftarrow$ empty path list
    $O_1, O_2 \leftarrow$ empty ontology providers
    $T_1, T_2 \leftarrow$ empty BFS expansion trees
    $R \leftarrow$ empty shared resource
    $C_{paths} \leftarrow$ initialize cache provide for paths
    if $C_{paths}$ already contains most semantic path between $v_1$ and $v_2$ then
        $P \leftarrow getPathFromCache(C_{paths}, v_1, v_2)$
    else
        $O_1 \leftarrow findOntologyConnectionSettings(c_1)$
        $O_2 \leftarrow findOntologyConnectionSettings(c_2)$

        $R \leftarrow$ initialize shared resource

        $T_1 \leftarrow bidirectionalSearch(v_1, O_1, R, L, K)$
        $T_2 \leftarrow bidirectionalSearch(v_2, O_2, R, L, K)$

        wait for both sides of bidirectional search

        $P_{all} \leftarrow extractPaths(T_1, T_2)$
        $P \leftarrow calculateMostSemanticPath(P_{all})$

        add founded most semantic path $P$ into cache $C_{paths}$
    end if
    return $P$
end function
```

Algorithm 4.2 Bidirectional search algorithm.

```
function BIRECTIONALSEARCH(v, O, R, L, K)
Input: $v$ is the root concept of expansion, $O$ is the ontology where $v$ resides, $R$ is the shared resource which is used by two sides of bidirectional search, $L$ is the maximum triple length to search possible paths, $K$ is the maximum number of shortest paths to search for most related semantic path (top-k shortest path)
    $Q \leftarrow$ empty queue
    $T \leftarrow$ empty tree
    $v_{next} \leftarrow$ null
    $p \leftarrow$ null
    push $v$ to queue $Q$ as first element
    append $v$ to shared resource $R$
    while $Q$ is not empty do
        $v_n ext$ pop next item from $Q$
        append $v_{next}$ to shared resource $R$

        $p \leftarrow getExpandedPath(v, v_{next})$

        if $p$ is a valid semantic path according to Hirst and St-Orge then
            if $length(p) \leq L$ then
                if shared resource $R$ contains $v_{next}$ then
                    append $p$ to $T$
                end if
                expand $v_{next}$ neighbour nodes and push into $Q$
            end if
        end if
    end while
    return $T$
end function
```

4.3.1.1 Adaptation of bidirectional breadth-first search

Breadth-first search (BFS) is an example of uninformed search algorithms. Uninformed search algorithms do not have any additional information beyond the problem definition and try to reach the target by expanding the nodes on the path. These algorithms are classified according to the strategy it uses to expand the next child node.

Starting with the root node (or with a given initial node), BFS expands child nodes of each node to reach the target node, and it does not pass to lower level until all child nodes are expanded. If there is a solution, BFS guarantees that it can be reached. Figure 4.6 shows how a tree can be expanded with BFS approach starting from the root. These steps are as follows:

- In the first step, the root node is expanded and the nodes marked with number 2 are obtained,
- In the second step, node 2 on the left is expanded and the nodes marked with number 3 are obtained,
- Node 2 in the middle is expanded and node 4 is obtained,
- Node 2 on the right is expanded and nodes marked with number 5 are obtained,
- The expansion process continues with this approach until the result is found.

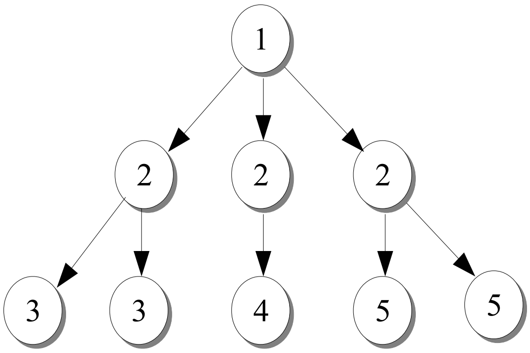


Figure 4.6: An example of BFS algorithm.

If it is assumed that the tree is uniform and each node has b successors, the first step will generate b nodes. Since each of these b nodes have also b successors, a total of b^2 nodes will be generated in step 2, b^3nodes in step 3, and so on. Assuming that the solution has been found at depth d, the worst case complexity will be $O(b+b^2+b^3+...+b^d)$ and hence $O(b^d)$.

To reduce the cost of BFS, the search can be performed simultaneously from two ends (from initial node and goal) hoping that both searches will meet in the middle. This search method is called bidirectional BFS and have $O(b^{d/2})$ complexity. Figure 4.7 shows an example where the node marked with yellow is the initial node and the node marked with blue is the goal. In this case, the search will be started simultaneously from both sides in the specified directions (to forward from initial node and to backward from goal).

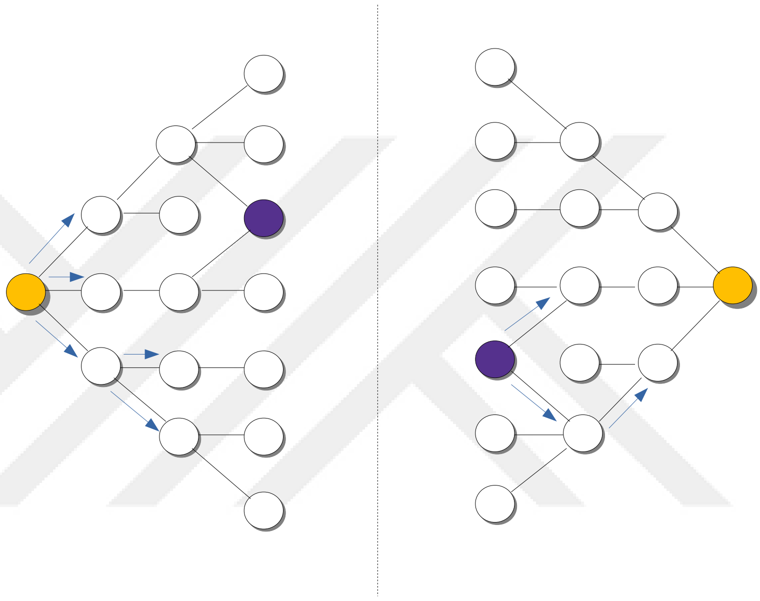


Figure 4.7: An example of bidirectional BFS algorithm.

Within the scope of this thesis, it is decided to use bidirectional BFS algorithm to find most related path between concepts. However, due to some technological constraints and requirements, some customization has been made in the implementation of this algorithm. Ontologies to be processed are huge data sources that contain hundreds of thousands of concepts and tens of millions of relationships. Therefore, these data sources need to be stored with specialized data solutions, such as ontology and/or graph databases, due to their performance and scalability needs. These database solutions do not have sufficient and customizable query capabilities to find the shortest path using semantic relatedness in an integrated manner. Therefore, the path finding method has been developed using the flow shown in Figure 4.8. First of all, using the two-way BFS algorithm, top-K shortest path between the concepts should be found. The semantic validity of each path is then checked and the relatedness values are calculated. The path with the highest relatedness value is accepted as the result. This method will not give an exact result. However, the semantic paths found would be acceptable for the reuse process. A similar solution is applied in the (Hulpuş et al., 2015) study, which aims to calculate the relatedness at the Abox level.

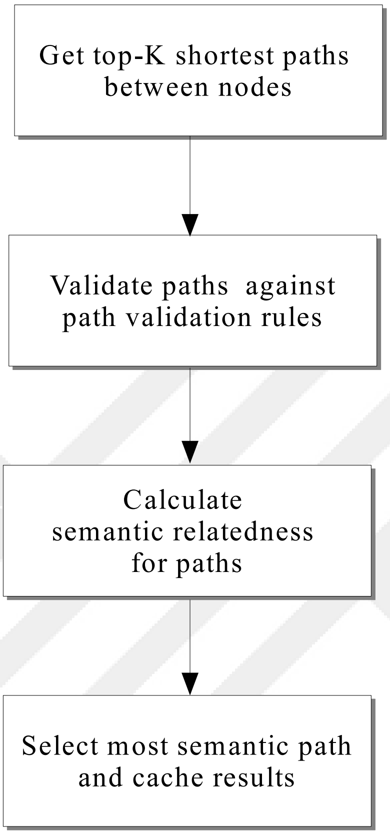

Get top-K shortest paths between nodes
Validate paths against path validation rules
Calculate semantic relatedness for paths
Select most semantic path and cache results

Figure 4.8: Path finding flow.

The pseudo-code of the two-way search algorithm is expressed in Algorithm 4.2. The $FindPath$ function, used to find paths between two concepts and takes parameters c_1, c_2, K, and L as input. c_1 and c_2 represent the concepts in which the path between them is to be found. The K parameter specifies the maximum number of shortest paths to be found between the two concepts, while the L parameter represents the maximum length of the valid paths. When calculating the length of a path, the triples along the path are taken into account. The default values are 10 for K and 20 for L. The values can be updated according to the selected ontology. The $FindPath$ procedure first accesses the cache to check whether the path between the two concepts already exists. If the relevant path is found in the cache, the search is terminated and the result is returned. Otherwise, the path search process starts with the steps listed below:

- The ontologies where the concepts are found are determined separately both for c_1 and c_2, and the connection strings of these ontologies are brought.
- The $bidirectionalSearch$ method is called for each concept to initiate simultaneous searches from both sides.
- Each running $bidirectionalSearch$ instance expands the successors according to BFS principles.
- At each expansion, the length of the existing path is checked. It is also examined whether it is a semantically correct path.
- If the existing path is valid, it should be checked for intersection with the search initiated from the other end.
- The found path is added to the list.
- The search continues until the K path is found or traversal is complete.

Communication between the two sides of search, which are running instances of $bidirectionalSearch$ method, is provided through a shared list that operates in accordance with the principles of concurrency. Both BFS instances mark the concepts they visit on this list and follow the concepts visited by the other end to find an intersection.

In the execution steps of path finding algorithm, the statement “semantically correct path” is firstly mentioned. This concept is very important in terms of adapting the path finding algorithms to ontologies and is detailed in the next subsection.

4.3.1.2 Semantically-correct paths

General purpose graphs can be traversed without any restrictions depending on whether they are directed or not. However, traversing semantic graphs (ontologies) without semantic constraints causes loss of meaning in paths. A solution to this problem has been proposed by Hirst and St-Orge (Hirst and St-Onge, 1998). Mazuel and his colleagues has also adapted this approach to calculate semantic relatedness and named as semantically correct path (Mazuel and Sabouret, 2008).

Hirst and St-Orge, in their work proposed to calculate similarity in WordNet, stated that traversing semantic graphs without any restrictions would cause loss of meaning. In order to define the restrictions they proposed, they divided the movement directions on WordNet into 3 groups:

- **Upward (U):** Hypernymy, moving from subclass to superclass.
- **Downward (D):** Hyponymy, moving from superclass to subclass.
- **Horizontal (H):** Meronymy and other relations.

They defined two rules in accordance with these definitions:

1. No other directions are allowed before an upward link. Once a downward or horizontal link has narrowed down the scope, it is not allowed to enlarge the scope again using an upward link.
2. Only one change of direction is allowed on the path. Because direction changes creates large semantic distances. However, horizontal relations are allowed to make a transition between the upward and downward links once.

To better understand the importance of these rules, a taxonomy section taken from WordNet is given as an example in Figure 4.9. If the semantically correct path constraints are not taken into account, the $Vehicle \rightarrow WheeledVehicle \rightarrow Container \rightarrow Bag$ path will be considered as valid. Because there are only 3 relationships on this path, the relatedness value is expected to be high. However, the direction changes have created large semantic gaps that cannot be determined by the relatednes calculation without using any traversal restriction.

Hirst and St-Onge listed the paths that may be valid according to the constraints they define, as indicated in Figure 4.10. All other paths are considered semantically invalid.

Each direction in a path is allowed to repeat several times in succession. For example; a path in the form of $U \rightarrow U \rightarrow U \rightarrow H \rightarrow D$ can be reduced to $U \rightarrow H \rightarrow D$. An example showing this situtation is illustrated in Figure 4.11.

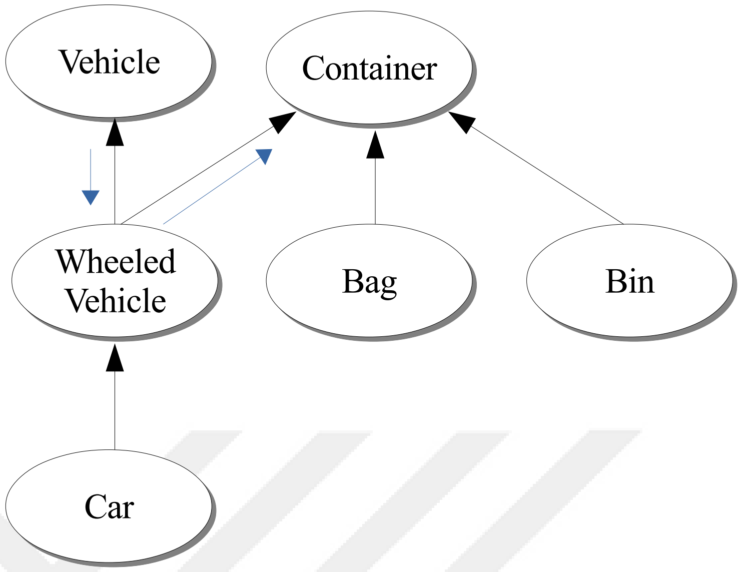


Figure 4.9: A path sample without semantically correct path restrictions.

The study proposed by Hirst and St-Onge has high importance within the scope of this thesis and should be integrated into the path finding method. However, some works have to be carried out within the scope of integration. Hirst and St-Onge defined the directions by using WordNet relations. OWL ontologies have much more relation types and these directions should be adapted to OWL ontologies. In this context, the reduction work which is stated in Table 4.2 has been carried out. Constructs not included in this table have either no effect on the path or are invalid, as will be specified in the following sections.

Table 4.2: Adapting path directions to OWL constructs.

OWL construct	**Corresponding direction**
rdfs: subclassOf (Generalization)	Upward
rdfs: subclassOf (Specialization)	Downward
owl:intersectionOf	Upward
owl:unionOf	Downward
rdfs:domain, rdfs:range	Horizontal
owl:allValuesFrom, owl:someValuesFrom	Horizontal
owl: equivalentClass	Horizontal

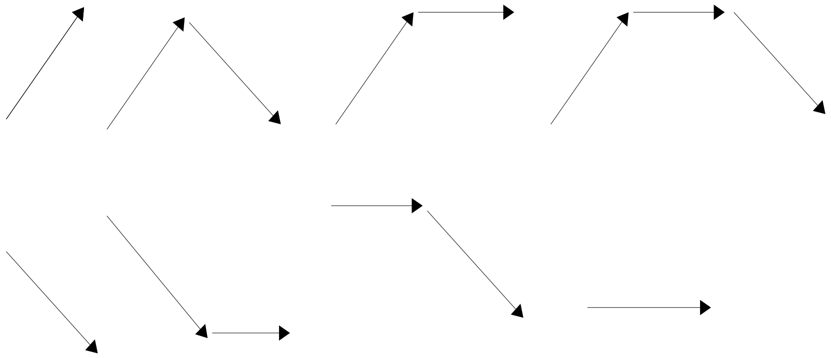


Figure 4.10: Semantically correct paths (Hirst and St-Onge, 1995)

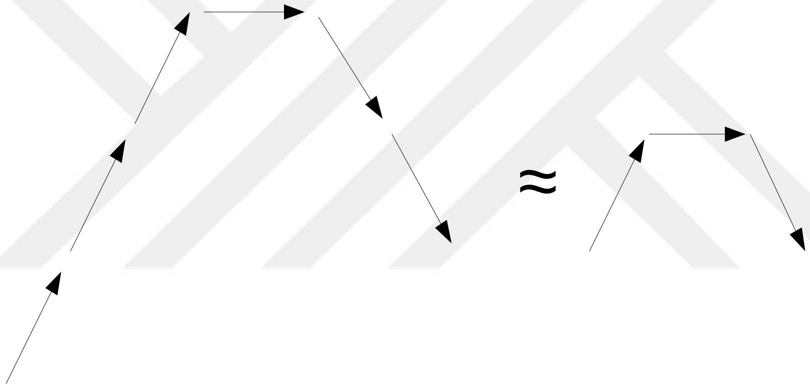


Figure 4.11: Recurrence of directions in a path.

4.3.1.3 Calculating semantic relatedness

In order to calculate the semantic relatedness values of the shortest paths found by the bidirectional BFS algorithm, OWL semantic relatedness values proposed by Giray and Ünalir are used (Giray and Ünalır, 2013). The main reason for the selection of this study is that it includes nearly all OWL constructs and may give faster results since no additional calculation is required other than simple multiplication operations. The semantic relatedness values proposed by Giray and Ünalir are given in Table 4.3.

Cost increases as the path length increases in general purpose graphs. However, the semantic relatedness value should decrease as the path length

Table 4.3: OWL semantic relatedness values (Giray and Unalir, 2013)

OWL construct	Semantic relatedness value
inverseOf	0.85254
equivalentClass	0.80228
subClassOf (generalization)	0.72149
subPropertyOf (genealization)	0.70409
domain (from property to domain)	0.69768
subClassOf (specialization)	0.69384
equivalentProperty	0.68156
subPropertyOf (specialization)	0.66135
disjointWith	0.65774
domain (from domain to property)	0.58072
range (from property to range)	0.50690
range (from range to property)	0.30593
differentFrom	0.19194

increases in semantic graphs. Therefore, Giray and Ünalir stated that the path relatedness value should be calculated by multiplying the semantic relatedness coefficients of all OWL constructs on the path.

In order to understand better how semantic relatedness values are calculated, an ontology section from WordNet is given as an example on Figure 4.12. According to this definition, the semantic path found between Car and $Motorcycle$ is $Car - subClassOf(generalization) \rightarrow MotorVehicle - subClassOf(specialization) \rightarrow Motorcycle$. The defined semantic relatedness values for subClassOf are 0.72149 for generalization and 0.69384 for specialization. The multiplication of these two values gives the relatedness value between Car and $Motorcycle$ as 0.506.

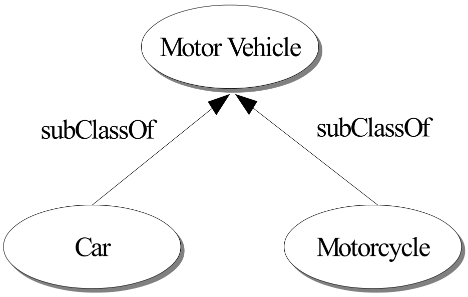


Figure 4.12: A WordNet sample to calculate semantic relatedness.

Although the semantic relatedness coefficients suggested by Giray and

Ünalir include most of the OWL constructs, there are still some missing OWL constructsmponents. For example; missing values such as intersectionOf and unionOf are frequently used in ontologies and should be included in the calculation. For this purpose, a study has been carried out to map the missing constructs to the existing values. The values obtained as a result of this study are given in Table 4.4.

Table 4.4: Reduction of missing components to existing relatedness values.

Missing component	**Reduction**	**Semantic relatedness value**
intersectionOf	subClassOf (generalization)	0.72149
unionOf	subClassOf (specialization)	0.69384
Restriction (onProperty)	domain (from domain to property)	0.58072
Restriction (allValuesFrom, someValuesFrom)	2*domain (from domain to property)*range (from property to range)	0.5887

4.3.1.4 RDF graphs and semantic relatedness

As already mentioned, it is possible to represent OWL ontologies as an RDF graph. However, the graph representation of ontologies constitutes some additional triples, and some assumptions and adjustments should be made during calculation of relatedness.

Figure 4.13 shows the OWL and the corresponding triple format of a concept from CPR[26] ontology. The difficulties that the RDF graph can constitute during the application of the relatedness values will be discussed using this example.

- **equivalentClass and anonymous classes:** Sometimes, it is required to use anonymous classes in OWL ontologies. This type of usage is usually observed with *equivalentClass* and *subClassOf.* For example, *medical-problem* class, which is shown in Figure 4.13, is defined as equivalent class of an anonymous class. Therefore, it is not possible to talk about a real equivalence definition. This kind of usage of equivalence

[26] https://bioportal.bioontology.org/ontologies/CPRO

OWL Format

```
<owl:Class rdf:about="medical-problem">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="etiologic-agent"/>
        <rdf:Description rdf:about="pathological-disposition"/>
        <rdf:Description rdf:about="pathological-process"/>
        <owl:Restriction>
          <owl:onProperty rdf:resource="representedBy"/>
          <owl:someValuesFrom rdf:resource="sign-recording"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="representedBy"/>
          <owl:someValuesFrom rdf:resource="symptom-recording"/>
        </owl:Restriction>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="Entity"/>
</owl:Class>
```

N-Triple Format

```
<medical-problem> <rdf:type> <owl:Class> .
<medical-problem> <owl:equivalentClass> _:gencprid51 .
_:gencprid51 <rdf:type> <owl:Class> .
_:gencprid51 <owl:unionOf> _:gencprid58 .
_:gencprid58 <rdf:type> <rdf:List> .
_:gencprid58 <rdf:first> <etiologic-agent> .
_:gencprid58 <rdf:rest> _:gencprid57 .
_:gencprid57 <rdf:type> <rdf:List> .
_:gencprid57 <rdf:first> <pathological-disposition> .
_:gencprid57 <rdf:rest> _:gencprid56 .
_:gencprid56 <rdf:type> <rdf:List> .
_:gencprid56 <rdf:first> <pathological-process> .
_:gencprid56 <rdf:rest> _:gencprid54 .
_:gencprid54 <rdf:type> <rdf:List> .
_:gencprid54 <rdf:first> _:gencprid55 .
_:gencprid55 <rdf:type> <owl:Restriction> .
_:gencprid55 <owl:onProperty> <representedBy> .
_:gencprid55 <owl:someValuesFrom> <sign-recording> .
_:gencprid54 <rdf:rest> _:gencprid52 .
_:gencprid52 <rdf:type> <rdf:List> .
_:gencprid52 <rdf:first> _:gencprid53 .
_:gencprid53 <rdf:type> <owl:Restriction> .
_:gencprid53 <owl:onProperty> <representedBy> .
_:gencprid53 <owl:someValuesFrom> <symptom-recording> .
_:gencprid52 <rdf:rest> <rdf:nil> .
<medical-problem> <rdfs:subClassOf> <bfo:Entity> .
```

Figure 4.13: OWL and N-Triple format of medical-problem concept (CPR ontology).

is omitted during semantic relatedness calculation and value of semantic relatedness is assumed to be 1.

- **subClassOf and anonymous classes:** As in the case of *equivalentClass*, the relatedness value of a *subClassOf* relationship defined using an anonymous class is also assumed to be 1.

- **domain-range relations:** The domain and range together defines relations among concepts. However, it is not possible to navigate between the concepts due to the directions of the domain and range relationship (See Figure 4.14). Therefore, domain-range constructs need special attention during traversal and they should be treated as one single horizontal relation. Otherwise, it is not possible to achieve the transition between two concepts.

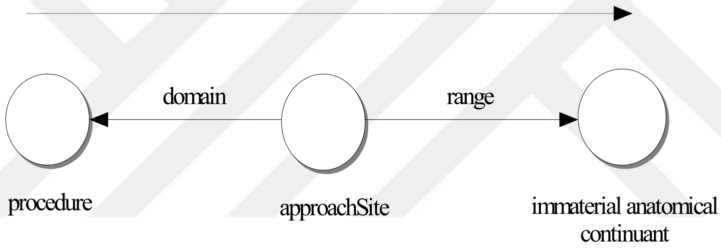


Figure 4.14: Directions of domain-range relations.

- **RDF lists:** OWL components such as *unionOf* and *intersectionOf* can define relationships using multiple classes. In the representation of these type of relations as N-Triple, RDF graph uses *rdf:list*. As shown in Figure 4.13, the relationship defined by *rdf:first* shows the first concept in the list and the relationship defined by *rdf:rest* shows the rest of the list with a sub-list. In such cases, the relatedness of the list-related components (*rdf: first* and *rdf: rest*) is ignored. It is assumed that the actual classes in the list are directly connected to *unionOf* and *intersectionOf* definitions.

- **Restriction:** *owl:Restriction* is a special form of anonymous class definition. OWL allows both of the definition of value and cardinality restrictions, but in this study, only value restrictions are considered. *onProperty* specifies on which property the restriction is to be applied. Therefore, it can be assumed that , if the related class has this property, this relation may be reduced to the domain definition. The *allValuesFrom* and *someValuesFrom* definitions give the allowed values which can be assigned to property specified with *onProperty*. Therefore, it is possible to reduce these relations to the range definition.

- **rdfs:label, rdfs:comment, owl:minCardinality ve owl:maxCardinality, etc:** The path finding algorithm only uses components whose domain and range are class types. Therefore, components with a range of string or integer such as *rdf:label, rdfs:comment, owl:minCardinality*, and *owl:maxCardinality* are excluded from traversal. Because, *label* and *com*ment only can hold string values, it is not possible to traverse graph using these constructs. The same applies for *minCardinality* and *maxCardinality* which have integer values.

4.3.1.5 A case study for path finding

In order to better understand how the path finding algorithm works, an example will be described in this section. The samples have been formed using CPR[27], OGMS[28], BFO[29], IAO[30], Biotop[31], Snomed-CT[32] and FMA[33] ontologies. Further details of this example dataset are provided in Section 7.2.

Two interrelated concepts were selected from the CPR: *sign-recording* and *clinical-act*. When the semantic path algorithm is run, it finds 5 valid paths (according to Hirst and St-Orge and path length restrictions) between these two concepts. Only the first three of these paths are shown in Figure 4.15.

The first two paths only traverse within CPR ontology. However, the third path crosses the borders of CPR ontology and traverse also BFO concepts to find a valid path, because CPR ontology reuses BFO ontology and our algorithm allows transitions between ontologies which reuse each other. Therefore, it is possible to say that semantic path algorithm may find paths between concepts even if they are not in same ontology.

[27] https://bioportal.bioontology.org/ontologies/CPRO
[28] https://bioportal.bioontology.org/ontologies/OGMS
[29] https://bioportal.bioontology.org/ontologies/BFO
[30] https://bioportal.bioontology.org/ontologies/IAO
[31] https://bioportal.bioontology.org/ontologies/BT
[32] https://bioportal.bioontology.org/ontologies/SNOMEDCT
[33] https://bioportal.bioontology.org/ontologies/FMA

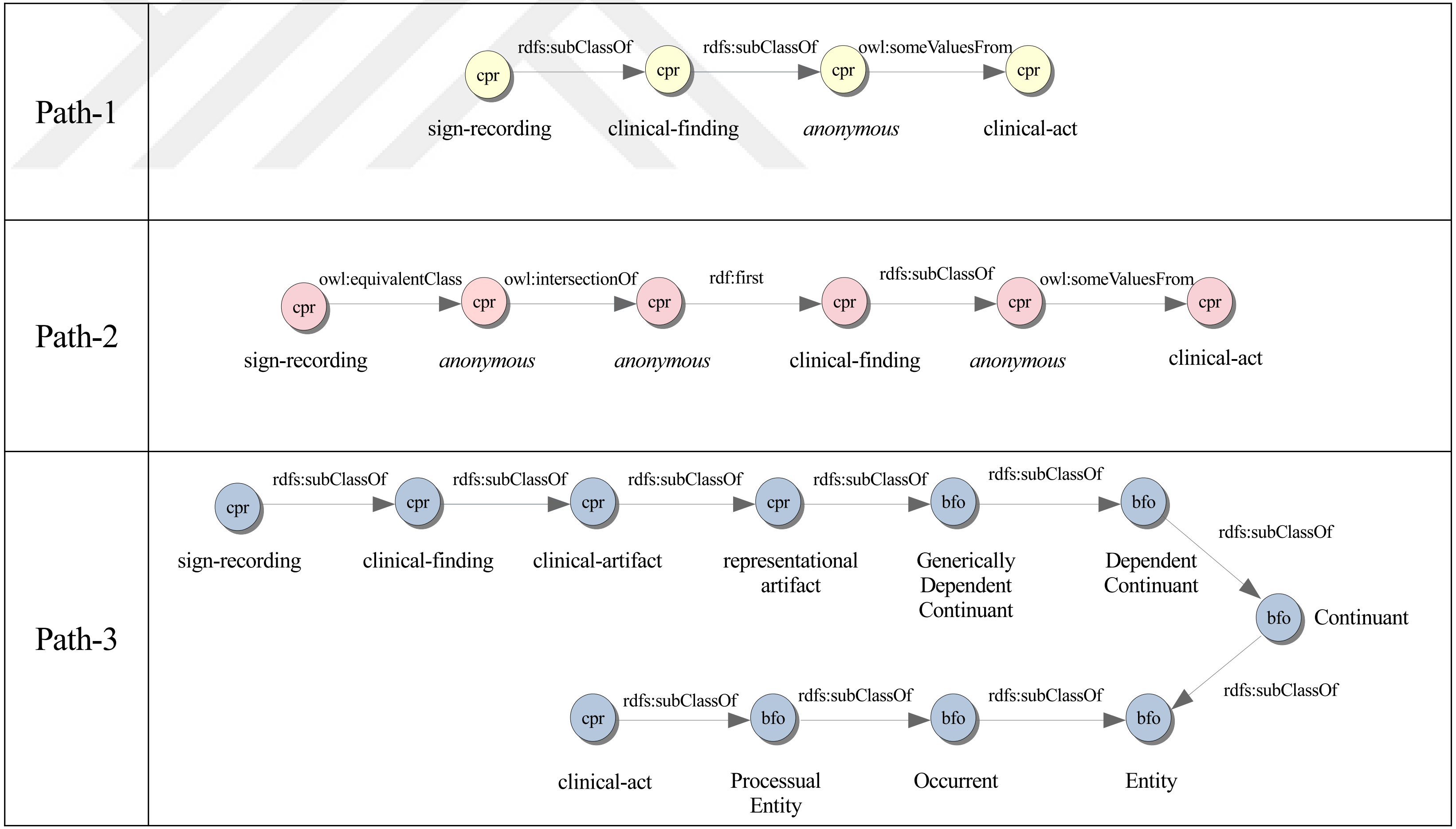


Figure 4.15: Top-3 shortest-paths between sign-recording and clinical-act entities

The next step after finding the shortest paths is to calculate the relatedness values for each path. Relatedness values of paths, which are calculated according to the semantic relatedness values and rules given in previous sections, are shown in Table 4.5 with details. As the path with number 3 is longer than the other paths, the relatedness value is also lower. However, although the lengths of the other two paths are different, the relatedness values are remarkably the same due to the existence of anonymous classes. Anonymous classes are directly related to the internal definition of the class, and in the calculation, the relatedness values of the relationships that are linked to these classes are assumed to be 1. Therefore, the relatedness values of the two paths are the same. However, because its length is shorter, Path-1 is chosen as the most semantic path.

Table 4.5: Semantic relatedness calculations of paths

Path	Relation	Relatedness value	Path relatedness value
Path-1	subClassOf	0,72149	0,4295
	subClassOf (anonymous)	1	
	someValuesFrom	0,5953	
Path-2	equivalentClass (anonymous)	1	0,4295
	intersectionOf	0,72149	
	first	1	
	subClassOf (anonymous)	1	
	someValuesFrom	0,5953	
Path-3	subClassOf	0,72149	0,03399
	subClassOf	0,72149	
	subClassOf	0,72149	
	subClassOf	0,72149	
	subClassOf	0,72149	
	subClassOf	0,72149	
	subClassOf	0,72149	
	subClassOf (reverse)	0,69384	
	subClassOf (reverse)	0,69384	
	subClassOf (reverse)	0,69384	

4.3.2 Ontology modularization algorithm

If two concepts are given as input, module extraction can be reduced to the problem of finding the most semantic path. However, if more than two concepts are selected, there is a need for a solution that integrates these concepts with the lowest cost and highest relatedness. This problem is called Steiner Tree problem in graph theory.

In this study, the ontology module extraction needs are reduced to the Steiner Tree problem and the selected concepts are integrated according to the highest relatedness values. Additions regarding the semantic completeness of the extracted ontology module are implemented according to the minimum completeness requirements in the literature (Courtot et al., 2011) (Hanna et al., 2012).

4.3.2.1 Steiner Tree problem

The Steiner tree problem aims to find the minimum sub-graph $G' = (V', E')$ that covers the selected vertices S from a given graph $G = (V, E)$, taking into account the cost function d between the edges. When the number of selected vertices is 2, the problem is reduced to the shortest path problem and when the number of selected vertices is equal to the number of vertices in the graph, it is reduced to the minimum spanning tree problem (Takahashi and Matsuyama, 1980).

Steiner Tree has a wide range of applications, and is widely used in biomedical and medical fields. For example, Sadeghi and Fröhlich (2013) attempt to find minimal sub-nets involving selected genes and proteins from biological networks. There are also important studies used to reveal pathways in cancer and protein relationships (Sun et al., 2017; Jahid and Ruan, 2012). In this context, it is seen that the Steiner Tree problem and its existing solutions may be useful in ontology modularization.

The Steiner tree problem is NP-Complete and there is no method to find the exact solution with polynomial time complexity. Approximation methods that try to find the closest solution instead of the exact solution can work with higher performance.

It is possible to divide Steiner Tree solutions into exact and approximate solutions. Approximation solutions that work with several heuristics can be classified in different ways. A classification is shown in Figure 4.16 (Voß, 1992). Some of the approximation methods mentioned in this classification are not included in the Figure 4.16 for simplicity.

There are many methods proposed in the literature for the exact solution of the Steiner Tree problem. The run-time complexity of these methods increases exponentially with the growth of the graph. Therefore, they offer a high cost with

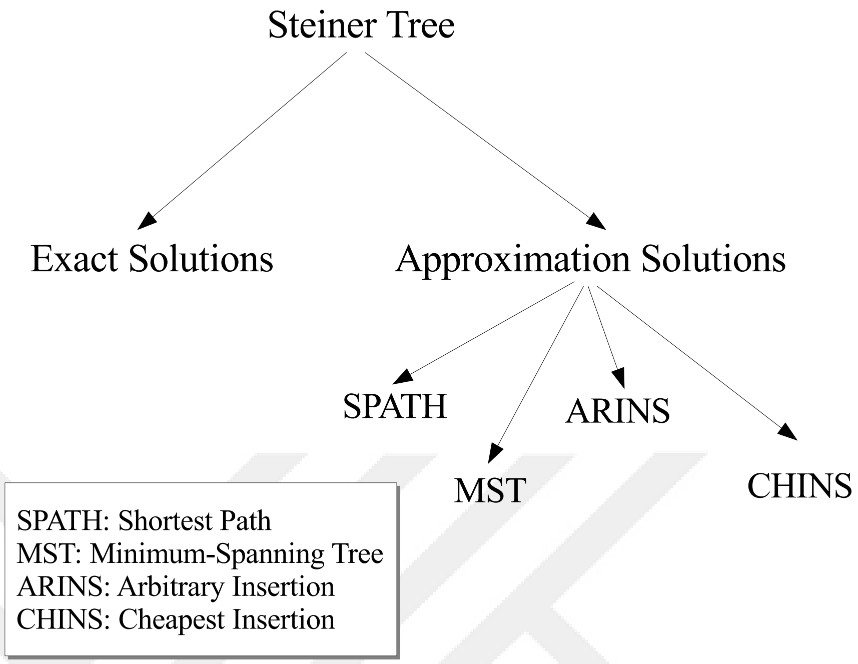


Figure 4.16: Classification of steiner tree solutions.

a exact solution. The aim of this thesis is to process very huge ontologies, such as SNOMED-CT and FMA which consist of hundreds of thousands of relationships. Therefore, exact solutions are beyond the scope of this study because of their run-time cost. However, some of important studies can be listed as Dreyfus and Wagner (1971), Levin (1971), Hakimi (1971), Balakrishnan and Patel (1987), and Beasley (1989).

On the other hand, algorithms that provide approximate solutions do not aim to find the exact Steiner Tree solution. However, the complexity of these algorithms is less than the exact solutions and are more suitable for purposes such as reuse. The approximation algorithms specified in the classification (See Figure 4.16) are detailed (Voß, 1992). Also, the notation and their meanings used while describing these algorithms are listed below (Voß, 1992):

- S: Selected Steiner vertices
- $G = (V, E)$: Original graph
- $G' = (V', E')$:Steiner tree solution

- $SP = (v_1, v_2)$:Shortest path between $v_1$and v_2
- l: leaf vertex

SPATH: Shortest Path Heuristic

1. Steiner Tree $G' = (\{n\}, Ø)$ is initialized with a randomly selected vertex from S
2. For every vertice $i = V - \{n\}$,shortest path $SP = (n, i)$ is found and added to G'

MST+P: Minimum Spanning Tree + Pruning

1. A minimum spanning tree $G' = (V', E')$ is created containing nodes in the set S
2. A leaf vertex l is selected and removed from solution recursively until meeting a Steiner vertex with its connected edges, if it is not in S.
3. Step 2 is repeated until there is no leaf vertex which is not in S

ARINS: Arbitrary Insertion

1. Steiner Tree $G' = (\{n\}, Ø)$ is initialized with a randomly selected vertice from S
2. A random vertex i is selected from remaining nodes $V'' = S - V'$ and shortest path $SP = (i, v')$ found between i and v',where $v' \in V'$. SP with minimum cost added to the solution G'
3. Step 2 is repeated until $S \subseteq V'$

CHINS: Cheapest Insertion

1. Steiner Tree $G' = (\{n\}, Ø)$ is initialized with a randomly selected vertice from S

2. All possible shortest paths between elements of remaning vertices $V'' = S - V'$ and $V'$is found. SP with minimum cost added to the solution G'
3. Step 2 is repeated until $S \subseteq V'$

CHINS-A: Cheapest Insertion (all root)

1. The CHINS algorithm is repeated for all possible starting vertices
2. The lowest cost sub-tree found is determined as the Steiner Tree solution

4.3.2.2 Selected Steiner Tree solution

The ontology module extraction method in this thesis is developed based on Steiner Tree. The aim is to adapt a Steiner Tree solution in the literature to the ontology module extraction process. Improving an existing Steiner Tree solution or proposing a new method is beyond the scope of this study. Therefore, the existing methods should be examined and evaluated according to the needs. The candidate Steiner Tree solution;

- should be path-based
- should be able to integrate semantic relatedness with a path-based solution
- should be easy to implement on graph technologies and graph databases (see Section 5).
- should be able to work with high performance. A fast working solution is more important for users than the most accurate but slower solution in the ontology reuse process. An approximation solution will not present a serious problem for ontology engineers.

These requirements are a starting point in determining the method to be selected and indicate that, minimum spanning tree or shortest path-based heuristic solutions may be appropriate. However, an evaluation study should also be performed among the candidate methods. When the comparison study performed by Vob is examined, it can easily seen that SPATH and MST + P solutions are not successful compared to other methods. But shortest path based methods which are classified as CHINS yield sufficient results. It also offers flexibility in terms of implementation and adaptability. The method and implementation proposed by Takahashi and Matsuyama (1980) are among the most prominent studies of this group.

Another classification and performance study has been conducted by Sadeghi and Förhlich (2013). As a result of the evaluations made in this study, the shortest path based methods yield satisfactory results. The authors also confirmed their assessment with the SteinerNet package[34], which was developed using the R programming language and the igraph[35] library. This package takes a graphml graph as input and performs evaluations according to the number of Steiner nodes determined. The methods they implemented in this package are listed below (The abbreviation for each group is given in the form the authors have stated).

Shortest-path approximation (SP)

For this group, the method proposed by Takahashi and Matsuyama was selected and implemented by authors. The sub-steps of this method are listed below:

1. Steiner Tree is initialized with a randomly selected node.
2. For every remaining Steiner node, which is not currently in Steiner Tree solution, and every node of current Steiner Tree solution, all possible shortest paths are found. Shortest path with minimum cost is added to Steiner Tree solution.
3. Step 2 is repeated until there is no Steiner node which is not in Steiner Tree solution.

Minimum-spanning tree approximation (KB)

Authors (Sadeghi and Fröhlich, 2013) have implemented their own method which is based on the method proposed by Kruskal to evaluate minimum spanning tree based solutions. The sub-steps of this method are as follows:

[34] https://github.com/cran/SteinerNet
[35] https://igraph.org/

1. Each Steiner terminal is considered as a seperate graph $f_{1,}f_{2,}f_3$ and so on.
2. The closest graphs to each other are merged sequentially. During merging, the shortest path between graphs are found for every possible of node pairs.
3. Step 2 is repeated until there is no remaining graph.

Randomized All Shortest Paths between Terminals (RSP)

This method has been proposed by also (Sadeghi and Fröhlich, 2013) authors. As first step, they constitutes a graph G^*,by connecting all shortest paths between Steiner nodes. Then, a minimum spanning tree $T$is extracted from G^*. Randomly selected non-terminal (Steiner) nodes v,where $v \in G^*$, is removed from G^*,unless $G^*$is divided into two connected components. Then, a minimum spanning tree extracted from $G^* - \{v\}$. If size of new spanning tree is smaller than T, removal of selected node is accepted. This procedure is repeated r times.

All shortest paths between terminals (ASP)

As stated by authors (Sadeghi and Fröhlich, 2013), this is a trivial method and included for only comparison reasons. As first step, all the shortest paths between terminals are computed. Then, all these shortest path are merged to form a graph.

Subgraph of merged steiner trees (STM)

This method is a modified version of shortest-path heuristic and implemented by authors (Sadeghi and Fröhlich, 2013). Instead of randomly selecting one of the Steiner nodes with equal shortest path to all nodes, all shortest paths are selected and added to sub-graph. At the end, sub-graph will contain several merged Steiner trees.

Sadegh and Förhlich indicate that the shortest path solutions provide better performance than the other solutions in their comparisons. They also emphasizes that the method proposed by Takahashi and Matsuyama is useful for many situations, especially when some performance improvements are made.

Although Sagech and Förhlich emphasize the suitability of the shortest path solutions, it is important to make this assessment on ontologies. For this reason, an evaluation study using the open source SteinerNet package has been carried out on our own data set. This data set contains five ontologies which are reused by Sleep Domain Ontology (SDO). Detailed information about Sleep Domain Ontology is given in Section 7.2. These ontologies used in Steiner Tree performance data set are listed below:

- CPR
- OGMS
- BFO
- IAO
- Biotop

These ontologies firstly transferred to Neo4j database using the Extract-Transform-Load (ETL) developed in thesis study. This ETL tool is detailed in Section 5.3. Using export features of Neo4j, imported data converted to graphml format. Graphml representation of these five ontologies contains 2332 vertices and 6298 edges. This dataset also detailed in Appendix-6.

Graphml dataset executed using the R code represented in Listing 4.1. *generate_st_samples* and *steiner_comparison_plots* functions provided by SteinerNet package (Sadeghi and Fröhlich, 2013). Using *generate_st_samples,* random data samples created, and for each sample performance evaluation executed.

```
install.packages(SteinerNet)

library(igraph)
library(SteinerNet)

g<-read_graph("sample_graph.graphml",format="graphml")

data_samples<-generate_st_samples(graph = g, ter_number =
 ↪  rep(3,20), prob = rep(0.5,20))
steiner_comparison_plots(type = c("ASP","RSP","SP", "KB", "SPM"),
 ↪  method = c("runtime"), data = data, outputname =
 ↪  "plot_3.pdf")
```

```
data_samples<-generate_st_samples(graph = g, ter_number =
 ↪  rep(5,20), prob = rep(0.5,20))
steiner_comparison_plots(type = c("ASP","RSP","SP", "KB", "SPM"),
 ↪  method = c("runtime"), data = data, outputname =
 ↪  "plot_5.pdf")

data_samples<-generate_st_samples(graph = g, ter_number =
 ↪  rep(8,20), prob = rep(0.5,20))
steiner_comparison_plots(type = c("ASP","RSP","SP", "KB", "SPM"),
 ↪  method = c("runtime"), data = data, outputname =
 ↪  "plot_8.pdf")

data_samples<-generate_st_samples(graph = g, ter_number =
 ↪  rep(10,20), prob = rep(0.5,20))
steiner_comparison_plots(type = c("ASP","RSP","SP", "KB", "SPM"),
 ↪  method = c("runtime"), data = data, outputname =
 ↪  "plot_10.pdf")
```

Listing 4.1: R code to evaluate Steiner Tree implementations.

As seen in Figure 4.17 and Figure 4.18, shortest-path based Steiner Tree approximations has also perform better results than other approximations on our datasets, and method of Takahashi and Matsuyama has been selected for ontology module extraction solution.

4.3.2.3 Steiner tree based module extraction

The Steiner tree solution, which is the core of the ontology module extraction method, is integrated with the path finding algorithm based on semantic relatedness. The pseudo-code of this module extraction method is given on Algorithm 4.3.

The first step of the *ExtractOntologyModule* procedure is to create the initial form of the module by identifying the concept with highest degree (sum of in-degree and out-degree) among the selected concepts. Thus, it is aimed to increase the possibility of connection with other concepts. The concepts visited in each cycle are stored in the *Visited* set. The outermost loop continues until the number of elements

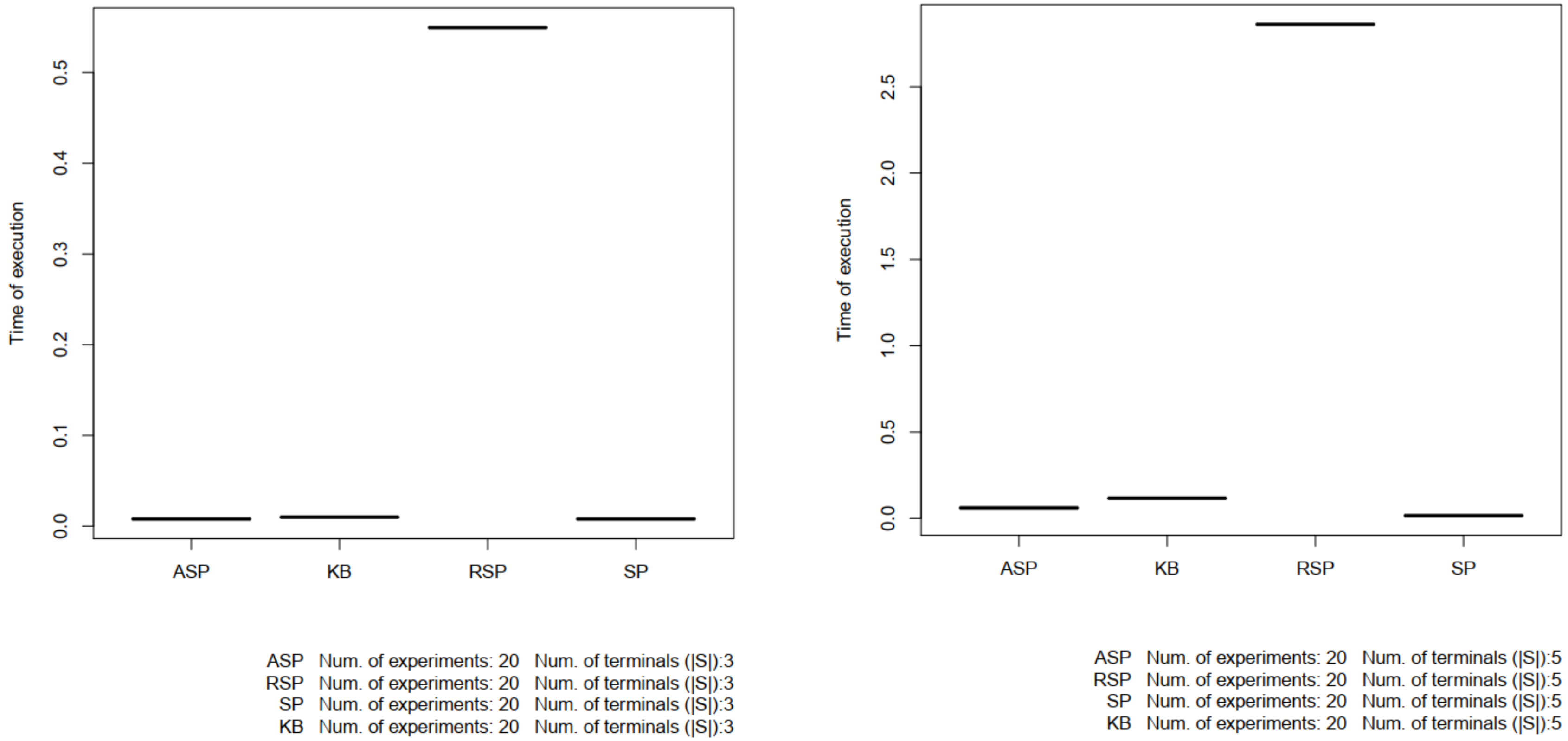


Figure 4.17: Steiner tree performance evaluation (3 concepts and 5 concepts).

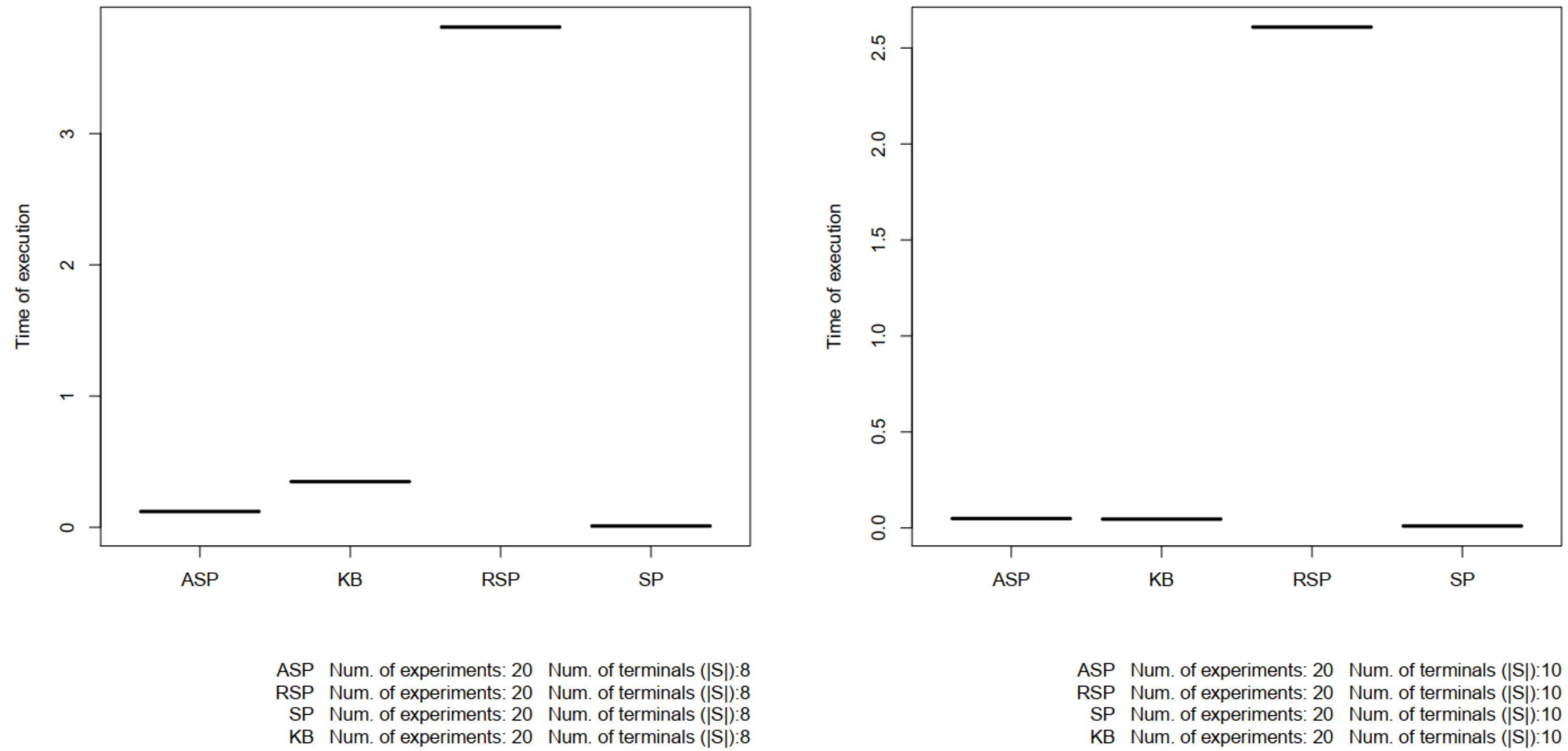


Figure 4.18: Steiner tree perfomance evaluation (8 concepts and 10 concepts).

Algorithm 4.3 Steiner tree based ontology modularization algorithm.

```
function ExtractOntologyModule($V_s$, $L$, $K$, $Params$)
Input: $V_s$ is the list of selected concepts from registered ontology sources, $L$ is the
    maximum triple length to search possible paths, $K$ is the maximum number
    of shortest paths to search for most related semantic path (top-k shortest path),
    $Params$ is user parameters
Output: $M$ extracted ontology module
    $Visited \leftarrow$ initialize visited concept list as empty set
    $M \leftarrow emptyontologymodule$
    $v_{temp}, P, P_{min} \leftarrow$ null
    $Pairs, P_{list} \leftarrow$ empty list

    repeat
        $v_{temp} \leftarrow$ select an unvisited node, usually the node with highest degree
        add $v_{temp}$ to Visited
        remove $v_{temp}$ from $V_s$

        $Pairs \leftarrow$ get list of every pair of nodes between $v_{temp}$ and $M$
        for each $pair \in Pairs$ do
            $P \leftarrow findPath(get(pair, 1), get(pair, 2), L, K)$
            add $P$ to $P_{list}$
        end for
        $P_{min} \leftarrow calculateMostSemanticPath(P_{list})$
        add $P_{min}$ to ontology module $M$
    until $length(Visited) = length(V_s)$
    $buildSyntacticValidation(M)$
    return $buildSemanticCompleteness(M, Params)$
end function
```

in the *Visited* set is equal to the number of Steiner nodes, in other words, until all Steiner nodes are processed. The first process in the loop is to find all concept pairs between existing Steiner Tree and unprocessed concepts. The most related path between each pair is then determined by using the path finding algorithm described in the previous section (There may not be a path between each pair of concepts). Among the paths found, the most related one is selected and added to the Steiner Tree set. This process continues until all selected concepts are processed.

During the adaptation of Takahashi and Matsuyama algorithm to find ontology modules, also some performance improvements have been performed. The first and most important improvement is the addition of cache support to the path finding process. If a previously searched path exists on the cache, results can be found without searching. The *ExtractOntologyModule* also performs path searches

Algorithm 4.4 Semantic completeness algorithm.

```
function buildSemanticCompletess(M, Params)
Input: $M$ is the sub-ontology extracted from original ontologies, $Params$ is the user paramers
Output: $M_c$ semantically completed ontology module
    $M_c \leftarrow M$
    ensure for missing parts
    if include all superclasses recursively is set true in $Params$ then
        $M_c \leftarrow appendAllSuperClassesRecursively(M_c)$
    end if
    if include all properties of classes is set true in $Params$ then
        $M_c \leftarrow appendAllProperties(M_c)$
    end if
    if include all equivalent classes is set true in $Params$ then
        $M_c \leftarrow appendAllEquivalentClasses(M_c)$
    end if
    if include all domain-range definitions of properties is set true in $Params$ then
        $M_c \leftarrow appendAllDomainRangeDefinitions(M_c)$
    end if
    return $M_c$
end function
```

of concept pairs simultaneously using threads. The maximum number of searches (maximum number of threads) that can be performed simultaneously is determined by the application parameters.

While the ontology module extracted by Steiner tree is core of the intended result, it may have some syntactic and semantic incompleteness. *buildSyntacticValidation* and *buildSemanticCompleteness* make additions that will make the ontology module syntactically and semantically valid. Details of these two methods will be discussed in next sections.

4.3.2.4 An example for module extraction

In order to better understand the ontology module extraction process, an example of 3 selected concepts is given step by step in this section. These concepts are selected from CPR ontology as path example and listed below:

```
http://purl.org/cpr/sign-recording
http://purl.org/cpr/clinical-act
http://purl.org/cpr/patient
```

As soon as the ontology module extraction algorithm is started, it chooses the initial concept. The initial concept is selected as the one with the highest degree (in-degree and out-degree), it other words the node which has the most connections. When the degrees of the selected concepts are calculated, it is seen that both the *sign-recording* and *clinical-act* have a degree of 10. In the case of equality, *sign-recording* is chosen as the initial concept since it comes first in parameter list.

Table 4.6: Degrees of selected concepts.

Class URI	**Degree**
http://purl.org/cpr/sign-recording	10
http://purl.org/cpr/clinical-act	10
http://purl.org/cpr/patient	6

After the selection of the initial concept, the ontology module became a set with a single element, as shown in Figure 4.19.

The next step is to find the shortest paths between the remaining selected concepts and the current concepts of the ontology module, using all possible pairs. In this example, the most semantic path between the two remaining concepts (*clinical-act* and *patient*) and *sign-recording* is the path between *sign-recording* and *clinical-act*. This path has a relatedness value of 0.4248 (see Figure 4.20).

After this stage, the ontology module is in the form of a path of 4 nodes (see Figure 4.20). The final step is to find the most meaningful path between 4 nodes and *patient*. This path is found between *clinical-act* and *patient* with a semantic relatedness value of 0.5887. The triples of this path are listed below and the current situation is shown in Figure 4.21.

With this step, the Steiner Tree algorithm has completed the process of extracting draft module. However, this module is not syntactically and semantically

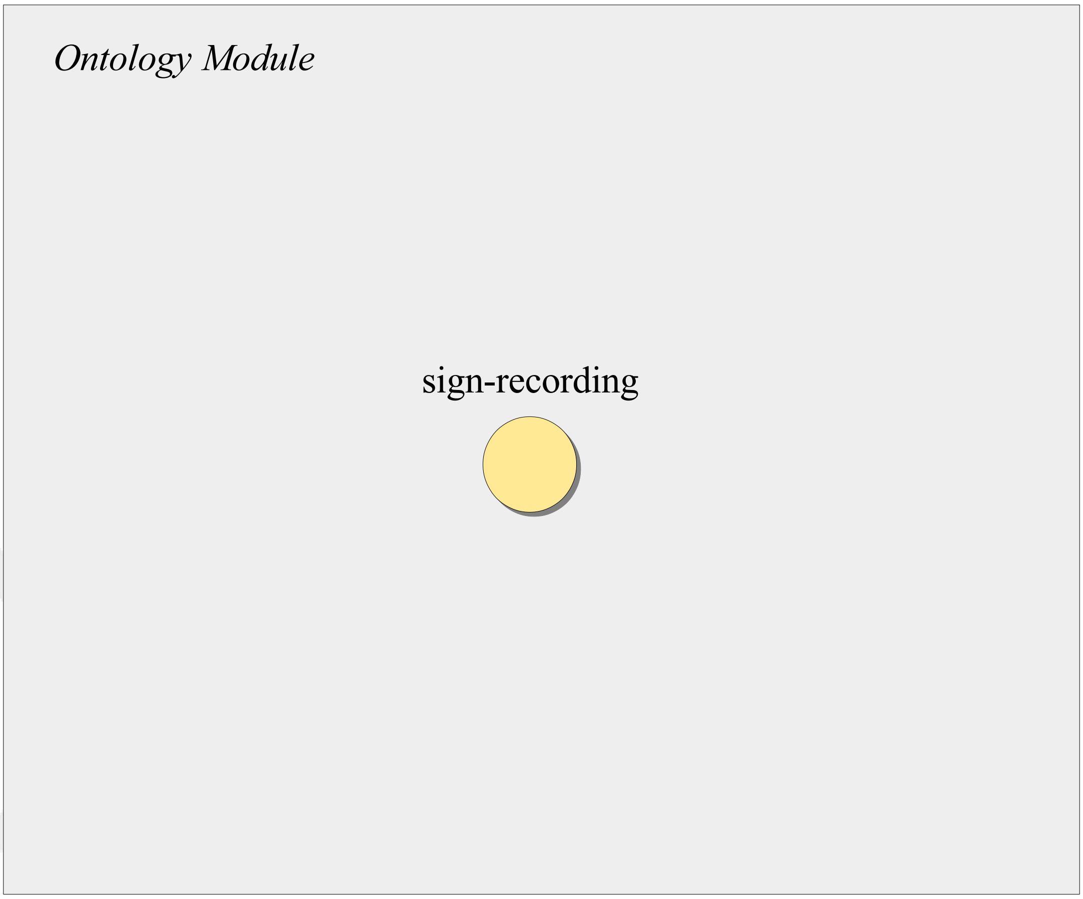


Figure 4.19: Ontology module extraction (Step-1).

valid and cannot be used as an ontology. In the next section, required activities to make this draft module syntactically and semantically valid ontology is explained.

4.3.2.5 Syntactic completeness

The path finding algorithm only deals with concepts and relationships between selected concepts. It does not take into account the syntactic completeness of the components along the path. However, a module that is not syntactically valid cannot be used as part of an ontology.

There is no a clear definition about the syntactic validity of ontologies in the literature. However, in the MOVE method, Well-Formedness Optimization Scheme rules do similar work.

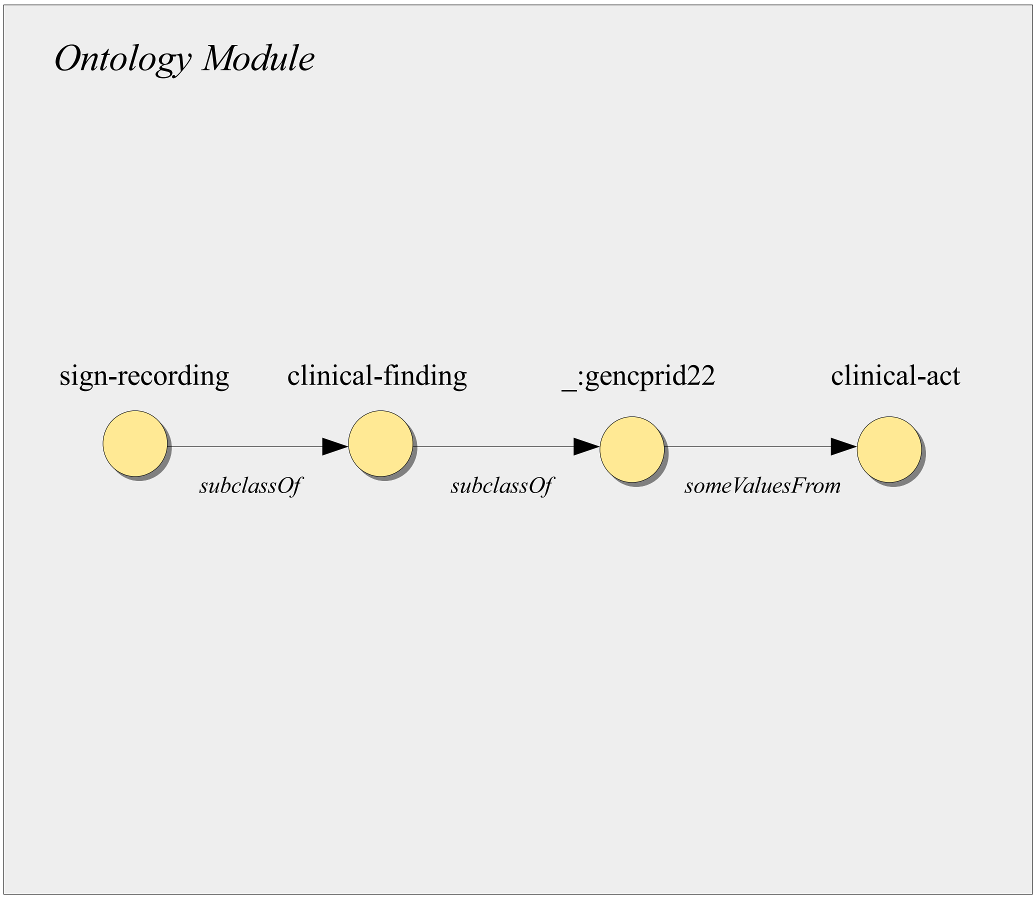


Figure 4.20: Ontology module extraction (Step-2)

To clarify the problem, let us take the ontology module given in Figure 4.21 as an example. This module includes 6 components. All these components are linked with *subClassOf* and *someValuesFrom* relations, but there is no clue about types of these components. Anonymous class is probably an instance of *owl:Restriction*, because it is connected to *someValuesFrom* relation. Therefore, it is not possible to consider this module as an ontology module before some syntactic corrections are made. Considering all possible scenarios, these corrections can be listed as follows:

- Types of all components (*rdf:type*)
- Missing parts of restrictions (*owl:onProperty*, *owl:someValuesFrom*, *owl: allValuesFrom,...*)
- Missing classes of intersections
- Missing classes of unions

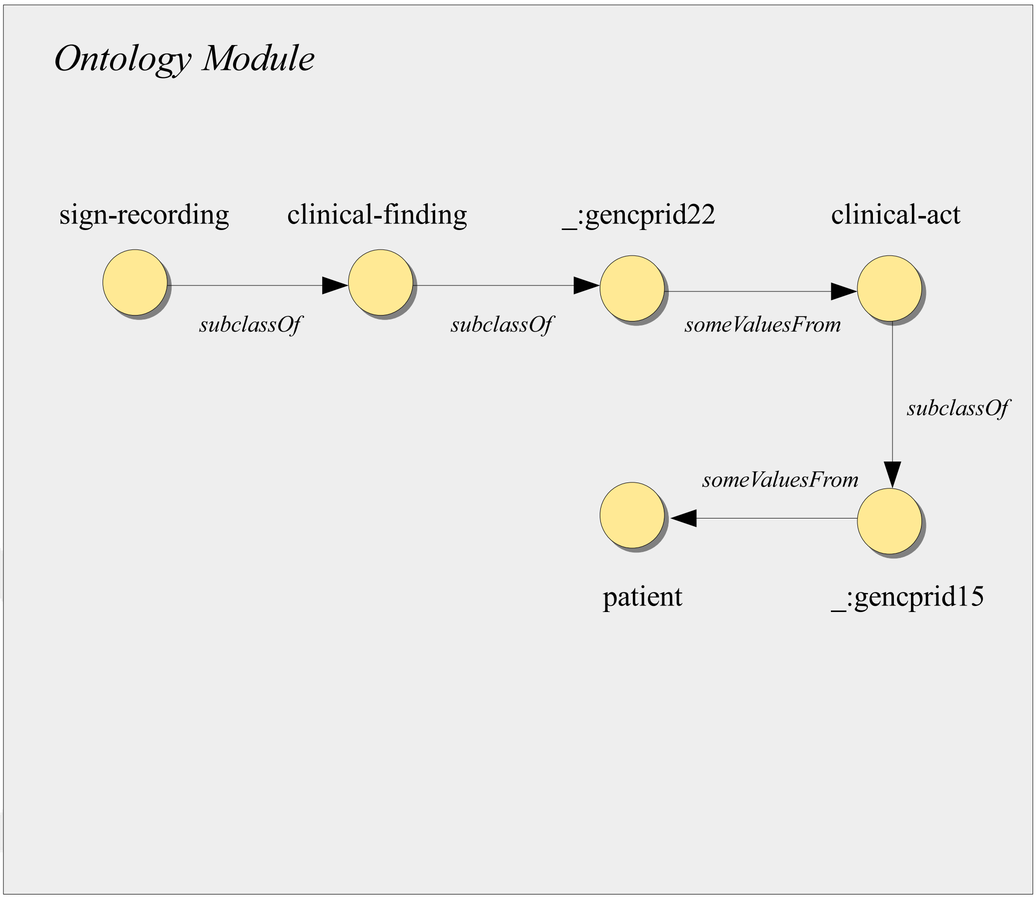


Figure 4.21: Ontology module extraction (Step-3).

The extracted example module in Section 4.3.2.4 is not a valid ontology module and can not be reused yet. As a first step, syntactic validation rules must be applied on this draft module. After applying these rules, module will be as shown in Figure 4.22.

The dotted components shown in the figure are subsequently added for validation. The relevant path is now syntactically valid and can be used as a module. If this module was expressed in XML format, it would look like in Listing 4.2.

```
<owl:Class rdf:about="clinical-act">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasParticipant" />
      <owl:someValuesFrom rdf:resource"patient"/>
    </owl:Restriction>
  </rdfs:subClassOf>
```

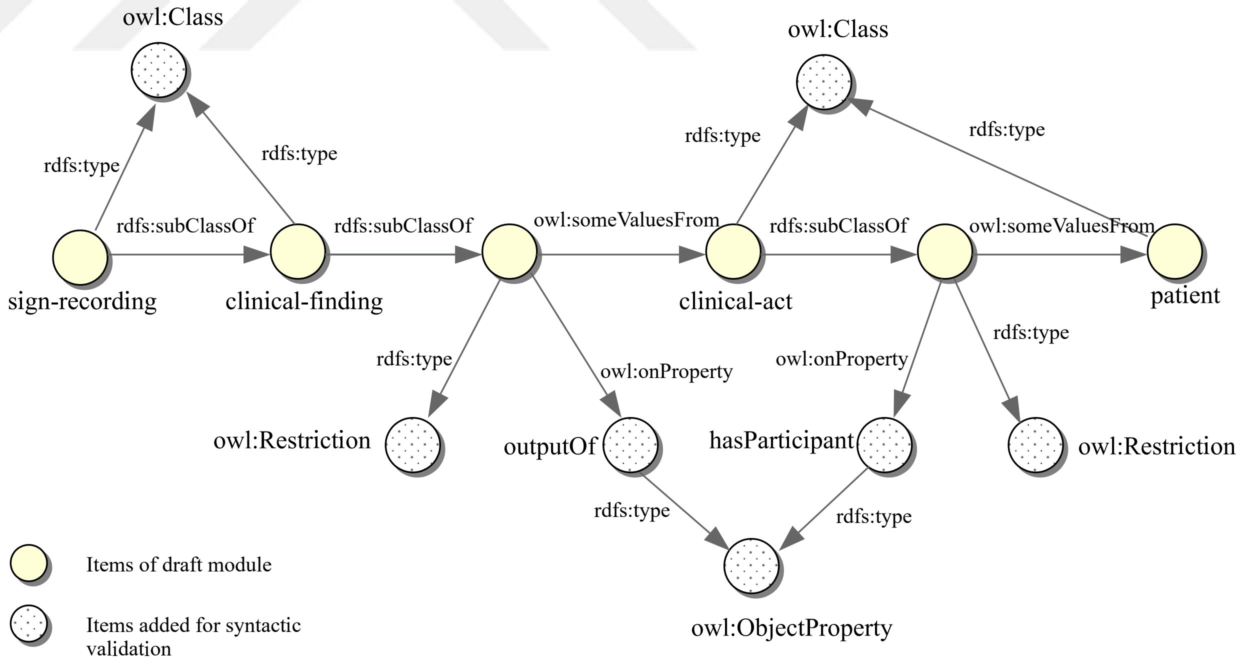


Figure 4.22: Extracted ontology module after syntactic completeness.

```
</owl:Class>
<owl:Class rdf:about="clinical-finding">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="outputOf" />
      <owl:someValuesFrom rdf:resource="clinical-act"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="sign-recording">
  <rdfs:subClassOf rdf:resource="clinical-finding" />
</owl:Class>
<owl:Class rdf:about="patient">
</owl:Class>
<owl:ObjectProperty rdf:about="outputOf">
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="hasParticipant">
</owl:ObjectProperty>
```

Listing 4.2: Extracted ontology module after syntactic completeness (RDF/XML format).

4.3.3 Semantic completeness

Syntactic completeness is the first step to ensure validity of the extracted ontology module. However, since it is not possible to define ontologies as a syntactic components implemented using a formal representation language, provision of semantic completeness should also be considered.

In the literature, especially in the field of ontology modularization, there are some important studies to ensure semantic completeness. One of these studies is the Semantic Completeness Optimization Schemes (SCOS) proposed within the MOVE project (Bhatt et al., 2006). SCOS requires that all ancestors, all part-of relations and all properties of concepts that are already selected for ontology module, must also be selected. Although these optimization schemes seem to be a necessity at first glance, they often cause the ontology module to grow beyond the needs and complicate it. Hussain and Abidi has pointed out this problem and stated that SCOS rules can cause the ontology module to be as large as the original ontology because of selecting all super-classes recursively

(Hussain and Abidi, 2009).

Recent studies suggest that it is not necessary to include so much information, such as recursive super-clasesses, all part-of relations, all properties, in order to maintain the semantic validity of the reused concepts, because URI of the imported term already refers to its location and other details. The most important and widely accepted of these studies is MIREOT principles (Courtot et al., 2011). MIREOT is abbrevation for *Minimum Information to Reference an External Ontology Term* and defines a set of rules to consistently import a term into new ontology. These rules are listed below:

- The URI of ontology containing external term
- The URI of the imported term
- The URI of the super-class in the target ontology
- Label and other textual definitions are recommended to ease the development of the ontology

The principles of MIREOT have been defined for the reuse of a single concept. However, the method developed within the scope of the thesis is concerned with the inclusion of an ontology module in the target ontology. Therefore, an additional study has been carried out to consistently include ontology module in target ontology, considering minimum information principles of MIREOT. All the other details about semantic completeness are left to the user in a parametric manner. These rules are detailed below:

- The concepts in the ontology module are divided into two groups as *core concepts* and *extended concepts*. Core concepts are concepts that are included in draft module (see Figure 4.22). Extended concepts are the other concepts that emerged during the execution of syntactic and semantic completeness processes.
- For extended concepts, only syntactic completeness must be carried out. There is no additional semantic completeness rule for extended concepts.
- All definitions of core concepts are included in the ontology module. These definitions are restricted to the only following relationships:

  - Equivalent classes
  - Direct super classes
  - Intersections
  - Unions
  - All other relations such as *disjointWith* are not included in ontology module

- All labels
- The following rules are left to the user with parameters:
  - Include all super-classes recursively
  - Include all equivalent classes
  - Include all properties of classes
  - Include all domain/range relations of properties

In this way, a minimum ontology module is created and this module can be expanded depending on the user's preference. Result of applying semantic completeness rules to ontology given in Listing 4.2 is shown in Listing 4.3.

```
<owl:Class rdf:about="clinical-act">
  <rdfs:subClassOf>
    <owl:Class rdf:about="ProcessualEntity"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasAgent"/>
      <owl:someValuesFrom>
        <owl:Restriction>
          <owl:onProperty rdf:resource="hasRole"/>
          <owl:someValuesFrom
           ↪  rdf:resource="healthcare-professional-role"/>
        </owl:Restriction>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasParticipant" />
      <owl:someValuesFrom rdf:resource="patient"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="clinical-finding">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="outputOf" />
      <owl:someValuesFrom rdf:resource="clinical-act"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="sign-recording">
  <owl:intersectionOf rdf:parseType="Collection">
    <rdf:Description rdf:about="clinical-finding"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="outputOf"/>
      <owl:someValuesFrom rdf:resource="clinical-examination"/>
    </owl:Restriction>
  </owl:intersectionOf>
  <rdfs:subClassOf rdf:resource="clinical-finding" />
</owl:Class>
<owl:Class rdf:about="patient">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="person"/>
          <owl:Restriction>
            <owl:onProperty rdf:resource="hasRole"/>
            <owl:someValuesFrom rdf:resource="patient-role"/>
          </owl:Restriction>
          <owl:Restriction>
            <owl:onProperty rdf:resource="participatesIn"/>
            <owl:someValuesFrom rdf:resource="clinical-act"/>
          </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="healthcare-professional-role">
</owl:Class>
<owl:Class rdf:about="clinical-examination">
</owl:Class>
<owl:Class rdf:about="patient-role">
</owl:Class>
```

```
<owl:Class rdf:about="ProcessualEntity">
</owl:Class>
<owl:ObjectProperty rdf:about="outputOf">
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="hasParticipant">
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="participatesIn">
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="hasRole">
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="hasAgent">
</owl:ObjectProperty>
```

Listing 4.3: Extracted ontology module after semantic completeness (RDF/XML format).

4.4 Conclusion

In this section, details of the semantic relatedness-based ontology modularization method that will form the basis of the reuse approach are given. Firstly, how the path finding algorithm is integrated with semantic relatedness is explained, then how the Steiner Tree problem is adapted to the minimum ontology module problem is detailed.

The most important feature of the ontology module extraction method proposed in this thesis study is that it tries to obtain the minimum valid ontology module using relatedness. It leaves the extension of the module of ontology to the users parametrically. In addition, thanks to its iterative structure, it can be easily integrated with reuse processes and tools.

This section detailed the ontology modularization method but does not give information about implementation. In next section, how this ontology modularization method and its underlying architecture implemented using semantic cloud architecture will be explained. Service and data architectures will also be detailed.

5. DEVELOPMENT OF SEMANTIC CLOUD ARCHITECTURE

One of the most important stages of the project developed within the scope of the thesis is the implementation of ontology modularization method on semantic cloud architecture. Semantic cloud architecture is not a clearly defined term in the semantic web and cloud literature and is a nomenclature that is considered appropriate in the context of this study. It means that ontology data sources and service components should benefit from the scalability and elasticity offered by cloud platforms using the appropriate solutions such as NoSQL databases, micro-service architectures, and so on.

The first requirement of compliance with cloud architecture is to have a scalable data model. Service components that will work on this data model should also be developed in accordance with the cloud-native (Gannon et al., 2017) approach (Abiş et al., 2018).

The architectural design of the server components developed to support ontology reuse is shown in Figure 5.1. Server components consist of data, modularization and REST service layers. Modularization layer (middle layer) contains modularization, metadata and search components. Also, a web interface and Protege plugin have been developed to make it easier to use by end users. End user components will be detailed in Section 6. This section only discusses the server components in details.

5.1 Design and Implementation of Semantic Data Model

A cloud-oriented application must first have an elastic and scalable data model. The elasticity of the data model is related to its ability to adapt to changes and updates (modifications). On the other hand, scalability refers to responding the requests by increasing application resources horizontally under heavy load. In addition, in our case, the data model is also expected to have semantic capabilities to store ontologies.

The data model of this application is based on the following basic requirements. It is also important to state that, this list describes the final version

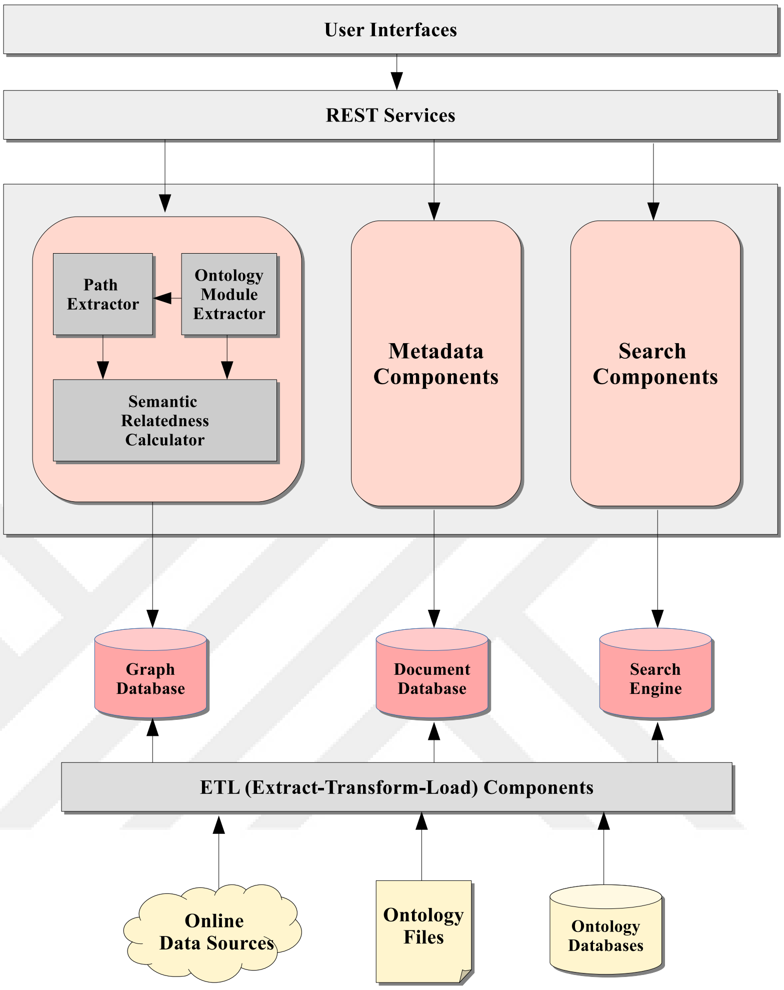


Figure 5.1: Semantic cloud architecture.

of the expected requirements and there have been so many changes and updates throughout the process.

- All parts of data model should be scalable and adaptable to possible changes.
- Ontology queries should be executed with high performance according to semantic relatedness values and semantically correct path constraints.
- Ontology data should be able to be traversed with high performance.
- Ontology data should be stored with a cache if necessary.
- Concepts in ontologies should be searchable with full-text queries.

- Basic information of ontologies (name of the ontology, usage area, dependencies, number of concepts, number of relationships, etc.) should be accessible quickly.
- The various log information obtained during the process should be stored and be accessible when necessary.

It is not possible to meet the above semantic data model requirements using a single data model such as relational, graph or document. Therefore, within the scope of the studies, a polyglot architecture (Sadalage and Fowler, 2013) consisting of 3 sub-data models has been proposed:

- Graph data model
- Full-text data model[36]
- Document data model

All technological and implementation details of these data models are detailed in the following subsections.

5.1.1 Graph data model

As mentioned in the previous sections, ontologies are RDF graphs, and a scalable and semantic graph data model is needed to store, query, and navigate these graphs. But, before creating this kind of data model, all requirements should be determined. However, it is not possible to clarify all of these requirements in the early phases of the thesis study. Therefore, in the first stages of the work, different technological solutions were tried, and the requirements and problems were revised incrementally. As a result, the final requirements of the graph data model were specified. It is possible to list these requirements as follows:

- Graph data model should be able to store ontology as an RDF graph without loosing semanticity.

[36] This term has been used to briefly indicate the data model to be created in search engines such as ElasticSearch and Solr, and there is no equivalent in literature.

- Graph data model should be able to scale (horizontal scaling) with high performance in semantic cloud architecture.
- Path traversal algorithms should be executed with high performance.
- Top-K shortest path between two concepts (or nodes) should be be found and listed using bidirectional BFS algorithm.
- Path traversal algorithms should be customizable and semantically correct path (Hirst and St-Onge, 1998) requirements should be adapted.
- Semantic relatedness calculations should be run on data while traversing graph, without having to modify internal structures.

The alternatives that are tried in the process of determining these requirements and final results are detailed in the following sub-sections. Also, details about implemented graph data model is given.

5.1.1.1 Accessing ontologies using SPARQL and REST

The first attempt at the development of the ontology graph model was to access ontology resources using SPARQL and other endpoints offered by providers, rather than designing a local storage solution. Thus, it was thought that ontologies could be integrated into the system quickly and storage costs can be reduced. However, the results of the experiments did not satisfy the expectations.

Ontologies such as Dbpedia, Wikidata, Yago and ontology repositories such as Bioportal offer SPARQL endpoints for remote access. In addition, there are projects such as Google Knowledge Graph and Microsoft Concept Graph which provides their own search interfaces (See Table 5.1). Although these endpoints are very useful for basic query requirements, complex traversal needs of our method does not fit into this solution.

Within the scope of the creation of the graph data model using remote endpoint approach, the architecture shown in Figure 5.2 has been designed and some parts of it has been implemented. As seen in the figure, considering the requirements of several data sources, such as Dbpedia, Yago, WordNet, Wikidata, Google Knowledge Graph and other local sources (data sources which do not have

Table 5.1: Remote ontology and knowledge sources.

Data Source	**Endpoint Type**	**Endpoint Url**
Dbpedia	SPARQL	https://dbpedia.org/sparql
Yago	SPARQL	https://linkeddata1.calcul.u-psud.fr/sparql
Wikidata	SPARQL	https://query.wikidata.org
Bioportal	SPARQL+REST	http://sparql.bioontology.org
Google Knowledge Graph	REST	https://kgsearch.googleapis.com
Microsoft Concept Graph	REST	https://concept.research.microsoft.com/api

query endpoints and are only downloadable as OWL files), a complex archiecture has been created.

Studies using remote access architecture have not yielded the expected results. The reasons for this are detailed below:

- Unpredictable network loads due to accessing remote resources.
- Unpredictable performance problems due to using public and uncontrollable endpoints used by multiple unknown users at the same time.
- Restrictions and precautions of ontology services against intensive use (Most of the ontology providers stop queries that cause heavy load and block IP addresses that send so many requests).
- Inadequacies of SPARQL query language in path traversals.
- Lack of customization and adaptation.
- Challenges of accessing multiple endpoints at the same time.
- For ontology resources without endpoints, it is still imperative to use a local data solution, and in this case, managing two separate technologies together brings complexity and problems.

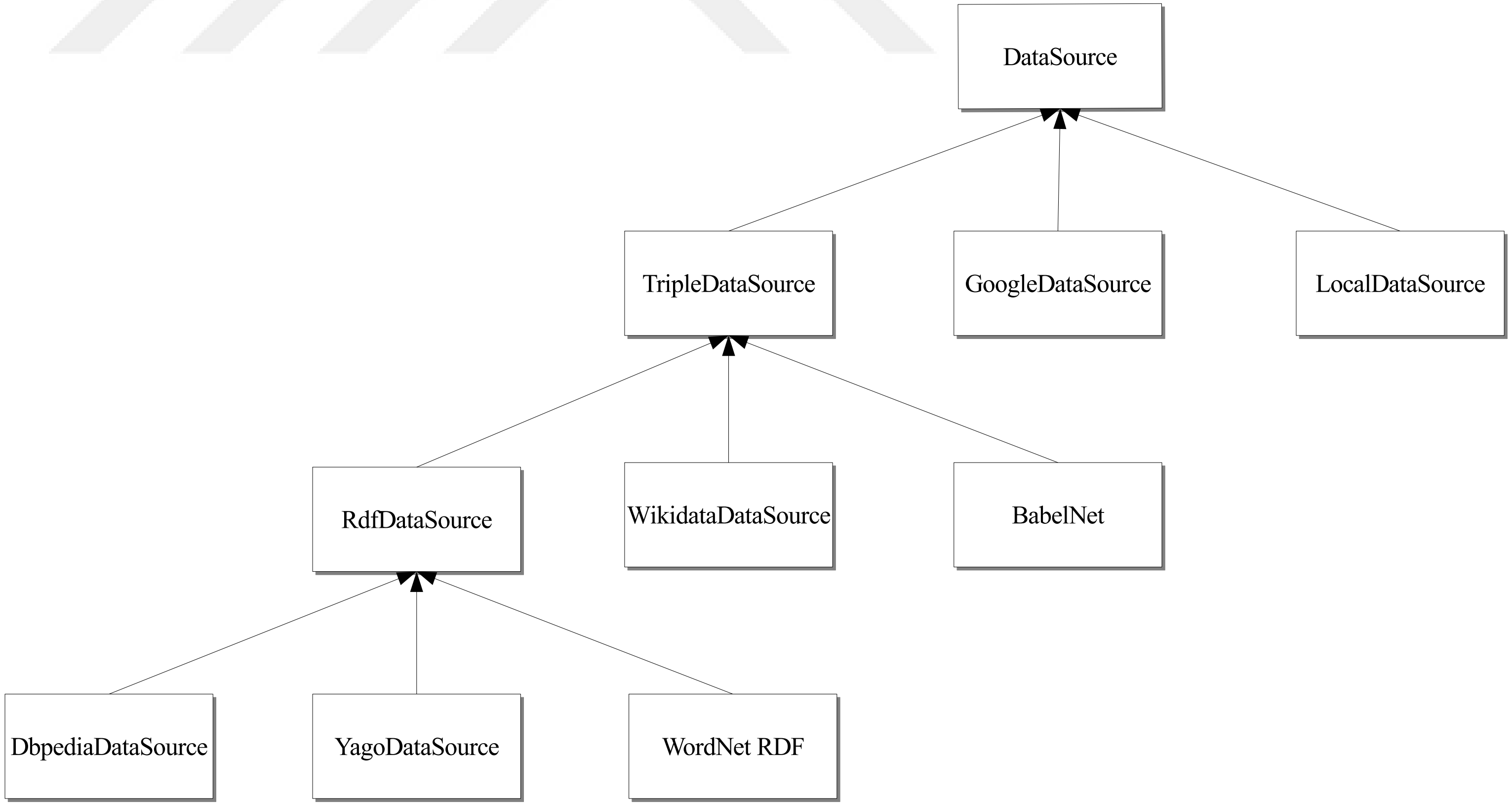


Figure 5.2: Architecture of graph data model for remote endpoint approach.

5.1.1.2 Using ontology databases

Since remote ontology resources are not suitable for our reuse method, studies have been performed to use ontology databases, which are special graph databases used to store ontologies. Although it is possible to process ontologies as local files using libraries such as Jena and OWL as an on-site solution, use of these storage solutions, called triple-store or ontology databases, is essential for performance, especially for large-scale ontologies.

The primary purposes of ontology databases are to store ontologies appropriately, to run SPARQL queries on it, and to perform inference operations. Ontology databases provide SPARQL as the primary query language. Although SPARQL is a comprehensive query language, it does not have the capacity to handle graph algorithms, such as path traversals. SPARQL only supports property paths (Harris and Seaborne, 2013) to find possible paths between selected nodes. Although property path queries can find the length of the path and the intermediate nodes along the path, other details such as relations are not directly accessible. Following SPARQL examples (see Listing 5.1) show how property queries work.

```
#Name of people known by Ahmet
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
{
  ?x foaf:mbox <mailto:ahmet@example> .
  ?x foaf:knows/foaf:name ?name .
}

#Name of people known by friends of Ahmet
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
{
  ?x foaf:mbox <mailto:ahmet@example> .
  ?x foaf:knows/foaf:knows/foaf:name ?name .
}

#All superclasses of x (recursively)
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema>
SELECT ?type
```

```
WHERE
{
  ?x rdf:type/rdfs:subClassOf* ?type .
}
```

Listing 5.1: Example property path queries.

We can clarify how SPARQL language uses property path queries to find paths between given nodes using a real example shown in Listing 5.2. This query aims to find whether a valid path exists between *Audi_A3*, an automobile model, and *MeanOfTransportation* class. *subClassOf* relation is used as a recursive property path statement. Then, to find the path length, *group by* operation is applied to query.

```
SELECT ?start ?end (count(?mid) as ?length)
WHERE {
        values (?start ?end) {(
         <http://dbpedia.org/resource/Audi_A3>
         <http://dbpedia.org/ontology/MeanOfTransportation>
    )}
    ?start rdf:type ?mid .
    ?mid rdfs:subClassOf* ?end .
}
group by ?start ?end
```

Listing 5.2: Relation between Audi_A3 and MeanOfTransportation class (Dbpedia).

The query in Listing 5.2 has been executed on Dbpedia ontology. Results of the query is shown in Table 5.2.

Table 5.2: Results of Dbpedia query.

start	**end**	**length**
<dbr:Audi_A3>[37]	<dbo:MeanOfTransportation>[38]	2

Although property path queries can find the length of the path and the intermediate nodes, other details such as relations are not directly accessible.

[37] dbr namespace is abbreviation for http://dbpedia.org/resource/

[38] dbo namespace is abbreviation for http://dbpedia.org/ontology/

Several additional round-trips has to be sent database to get relations of found path. These deficiencies of property path queries were also demonstrated at the challenges section of ESWC 2016 (ESWC2016, 2016) conference. To address these shortcomings, participants were expected to develop a SPARQL enhancement that could find the Top-K shortest path between two nodes in an RDF diagram. It is emphasized that this solution should return the paths entirely (including intermediate nodes and relations). Although some successful works have been performed within the scope of this challenge, such a feature has not been added to SPARQL language yet.

These limitations and inadequacies of SPARQL language are tried to be solved with special plugins provided by ontology databases such as Blazegraph (Blazegraph, 2017), Allegrograph (Allegrograph, 2020), Virtuoso (Virtuoso, 2020) and Stardog (Stardog, 2020). Throughout the thesis, various studies have been conducted in order to understand the suitability of these 4 databases to our method.

The Blazegraph database offers an infrastructure, called as Gatter-Apply-Scatter (GAS) (BlazegraphGASAPI, 2016), to integrate graph algorithms with SPARQL. Moreover, it can achieve higher performance by running these algorithms on the GPU (this feature is supported in commercial version). Although new graph algorithms can be developed with the GAS API, there are predefined and useful algorithms currently available:

- Bread-First Search
- Dijkstra
- Connected Components
- Page Rank

How Blazegraph GAS API components can be integrated with SPARQL is shown with a sample query in Listing 5.3. BFS is used as search algorithm in this example, with parameters maximum number of iterations and the maximum number of nodes to be visited.

```
PREFIX gas: <http://www.bigdata.com/rdf/gas#>
SELECT ?depth ?predecessor ?linkType ?out{
  SERVICE gas:service {
   gas:program gas:gasClass "com.bigdata.rdf.graph.analytics.BFS"
    ↪  .
   gas:program gas:in <http://dbpedia.org/resource/Turkey> .
   gas:program gas:target <http://dbpedia.org/ontology/Place> .
   gas:program gas:out ?out .
   gas:program gas:out1 ?depth .
   gas:program gas:out2 ?predecessor .
   gas:program gas:maxIterations 5 .
   gas:program gas:maxVisited 4000 .
 }
 FILTER (?depth>0)
 ?predecessor ?linkType ?out .
}
ORDER BY DESC(?depth)
LIMIT 100
```

Listing 5.3: Sample BFS query which uses Blazegraph GAS API on Dbpedia data.

In order to create a test environment, Dbpedia data has been transferred to the Blazegraph database running on our local servers, and the query shown in Listing 5.3 has been executed on Dbpedia data. The results of this query shown in Table 5.3.

Table 5.3: Results of Blazegraph GAS API query.

depth	**predecessor**	**linkType**	**out**
3	<dbo:PopulatedPlace>	<rdfs:subClassOf>	<dbo:Place>
2	<dbo:Country>	<rdfs:subClassOf>	<dbo:PopulatedPlace>
1	<dbo:Turkey>	<rdf:type>	<dbo:Country>

After using Blazegraph's GAS API, the first impressions were that these results could be sufficient for the path requirements of our study. In order to find semantic paths, it was planned to develop a relatedness based shortest path algorithm using GAS API. However, the implementation of the algorithm could not be performed due to the inadequacies and implementation costs of the Blazegraph libraries. Also, since there are no examples of this kind of usage, it is not possible to predict what kind of problems it will bring under heavy load, even if it is successful at the first stage. In addition, the integration of semantically correct path constraints would not be possible. Studies with the Allegrograph, Stardog and Virtuoso databases also failed to meet our semantic path requirements.

5.1.1.3 Using graph databases

Since accessing ontology resources via SPARQL endpoints and using ontology databases could not achieve the expected results, it is decided to use general purpose graph databases. They support graph analytics better and are more customizable. Because of its library capabilities, customizable structure and community support, Neo4j (Vukotic et al., 2014) has been chosen as the first alternative.

Neo4j is a native graph database that supports the property-graph model. It provides a ACID-compliant transactional data technology. The basic components of a Neo4j graph data model are nodes and relationships. Both nodes and relationships can have properties. Indexes, which are also called as labels, can be assigned to each node and relationship to group related constructs into sets and to increase query performance. All nodes indexed with the same label belongs to the same set. The same applies to relationships. Figure 5.3 illustrates an example of Neo4j graph data model.

Neo4j provides a comprehensive query language called CYPHER for graph queries and analytics (Vukotic et al., 2014). Since CYPHER is not used in this study, it will not be examined in detail in this report. However, some example CYPHER queries are given in Listing 5.4. These samples are taken from sample Neo4j database, *movie graph*[39].

```
//All Tom Hanks movies
MATCH (tom:Person {name: "Tom
 ↪  Hanks"})-[:ACTED_IN]->(tomHanksMovies)
RETURN tom,tomHanksMovies

//Movies and actors up to 4 "hops" away from Kevin Bacon
MATCH (bacon:Person {name:"Kevin Bacon"})-[*1..4]-(hollywood)
```

[39] https://neo4j.com/developer/guide-cypher-basics/

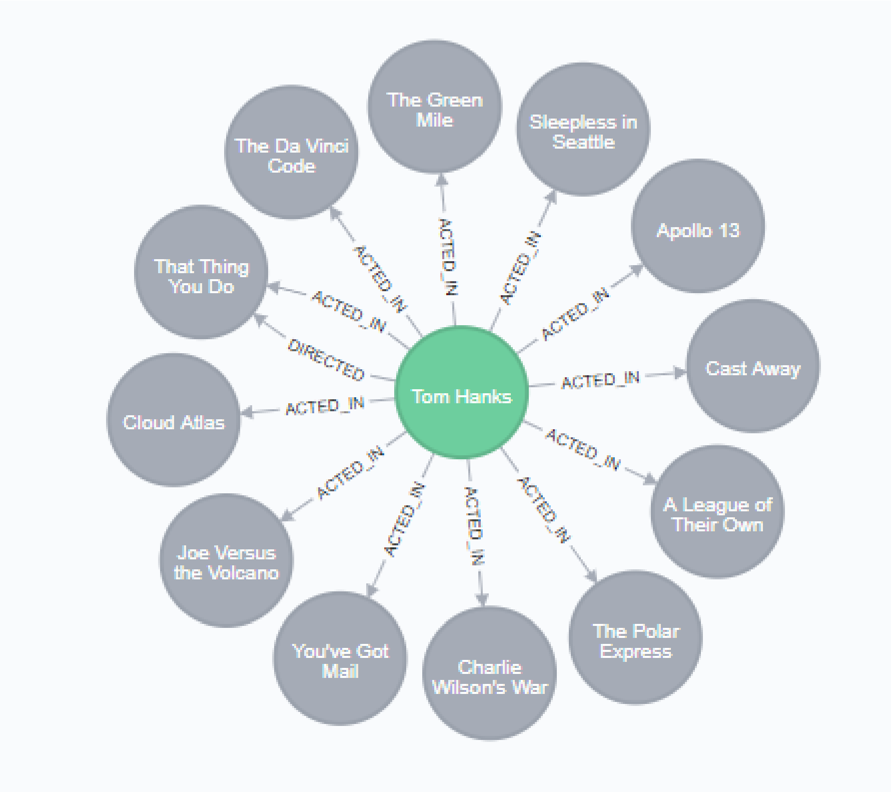


Figure 5.3: Neo4j graph data model sample.

```
RETURN DISTINCT hollywood

//Bacon path, the shortest path of any relationships to Meg Ryan
MATCH p=shortestPath(
        (bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person
        ↪  {name:"Meg Ryan"})
)
RETURN p
```

Listing 5.4: Sample Cypher queries.

Neo4j can be used in two different ways: server mode and embedded mode. In server mode, it operates as a standalone database server and access to the database is performed through CYPHER queries. In embedded mode, the database can run in the process of a separate application, in the same address space. In this approach, in addition to CYPHER queries, Neo4j libraries can be used directly, and complex operations and queries can be executed by the application. The differences between server mode and embedded mode are shown in Figure 5.4 (Vukotic et al., 2014).

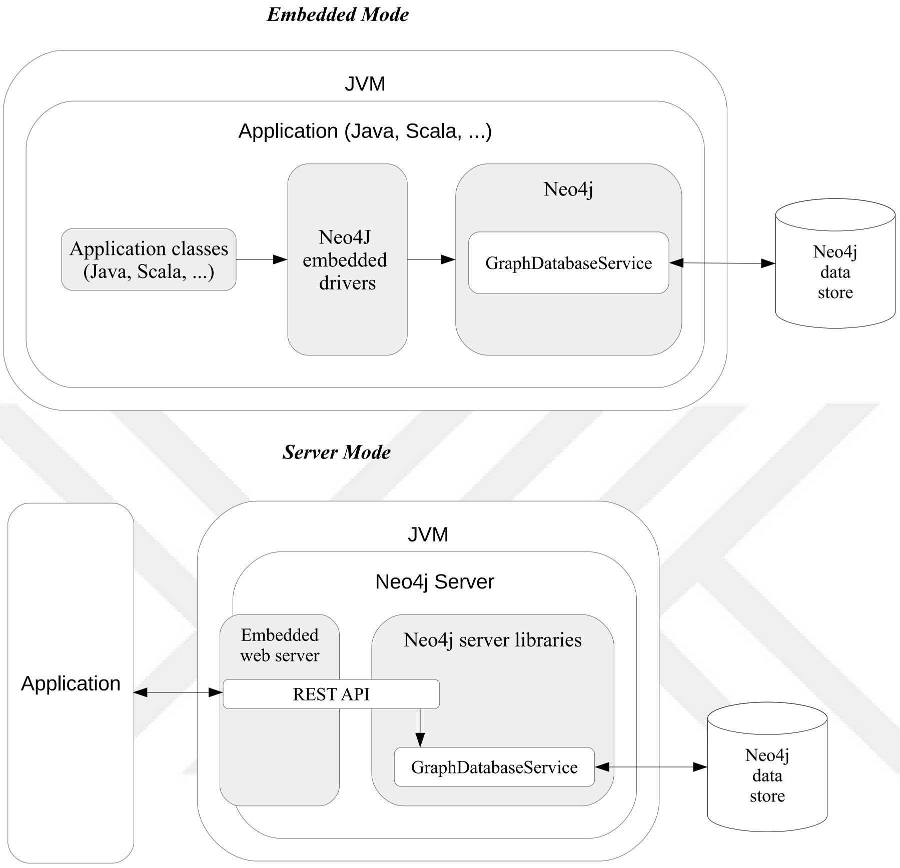


Figure 5.4: Neo4j modes (Vukotic et al., 2014).

In our work, embedded mode has been preferred in the implementation of the graph data model because it is more flexible, and more complex and customizable operations can be performed directly using Neo4j APIs.

It is not possible to store ontologies directly on Neo4j. Therefore, an appropriate data model design should be made with Neo4J constructs and data conversions from ontology to Neo4j should be performed in accordance with this model. In this context, first, a mapping between the ontology model and the Neo4J model has been defined (See Table 5.4). This mapping converts *rdf:type* statements into Neo4j labels. Therefore, classes and properties can easily be defined and indexed using label definitions. Properties with a range of primitive data types-*label*, *comment*, *minCardinality*, etc.-are not needed during traversal, so they are not transferred to the Neo4j data model. All OWL relations, such as *subClassOf*, *equivalentClass*, *unionOf* and *inverseOf*, are directly converted to Neo4J relations using appropriate labels.

Table 5.4: Mappings between OWL and Neo4j components.

OWL contructs		**Neo4j constructs**
type definitions (rdf:type)	->	Label
classes	->	node:CLASS
object properties	->	node:PROPERTY
data properties	->	N/A
relations (subClassOf, subPropertyOf, equivalentClass,....)	->	relation: SUB_CLASS_OF, relation:SUB_PROPERTY_OF,...
labels, comments	->	N/A
annotation properties	->	N/A

After performing this mapping and transfer, the appearance of a sample ontology on Neo4j is shown in Figure 5.5. The steps to complete this transfer and transformation are detailed with the ETL tool explained in Section 5.3.

5.1.2 Full-text data model

In order to support ontology reuse, a data model where ontology engineers can perform full-text searches within available resources should be provided. This data model should be elastic and scalable. For this purpose, different technologies were examined and evaluated. Although Apache Solr[40] has been preferred in the early stages, it has been decided that ElasticSeach would be the more suitable solution because of its capabilities and community support.

ElasticSearch is a comprehensive, distributed and widely used search engine and NoSQL document store based on Lucene library (Gheorghe et al., 2015). It stores data structures that have been serialized as JSON. It is highly scalable under heavy load and provides an elastic data model with its schema-less structure. All indexing and query operations can be performed using ElasticSearch REST APIs.

ElasticSearch basically consists of two components: Indices and documents.

[40] https://lucene.apache.org/solr/

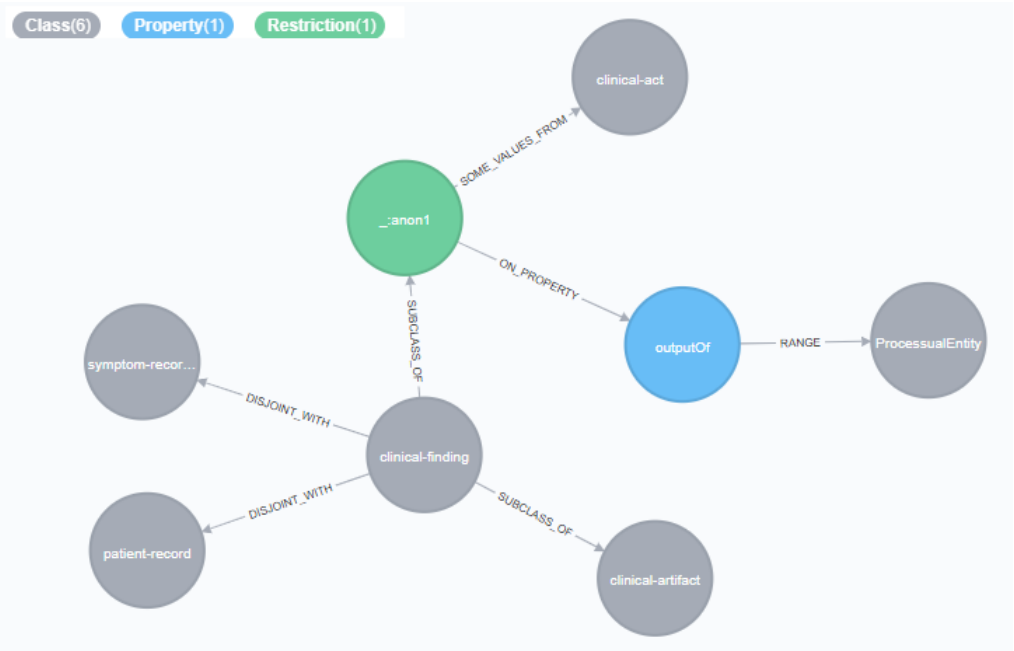


Figure 5.5: A sample ontology on Neo4j data model.

Indices are logical namespaces that contain multiple documents. It is possible to consider indices as ElasticSearch equivalent of the database instance in relational databases. Each index can contain multiple types to store documents. ElasticSearch data model is illustrated in Figure 5.6. ElasticSearch also contains administrative components such as clusters, shards and replicas, but they are not detailed in scope of this thesis study.

To provide a full-text search environment in scope of thesis study, a single ElasticSearch index and type has been created. This type only includes label definitions of ontology. Other annonation properties is omitted for simplicity. The mapping content used to define the ontology search index is given in Listing 5.5 as Json.

```
curl -X POST 'http://localhost:9200/sdo' -d \
'{
   "settings":{
     "analysis":{
      "analyzer":{
       "label_analyzer": {
        "tokenizer": "standard",
        "char_filter": ["label_filter"],
        "filter": ["lowercase"]
       }
```

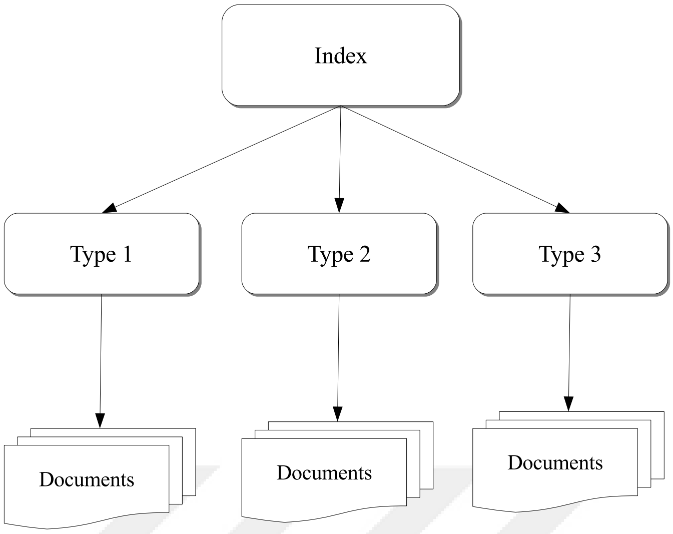


Figure 5.6: ElasticSearch data model.

```
    },
   "char_filter": {
     "label_filter": {
       "type": "pattern_replace",
       "pattern": "_|\\|.",
       "replacement": " "
     }
   }
  }
 },
"mappings": {
  "ont_class": {
    "properties": {
     "labels": {
      "type": "nested",
      "properties":{
        "label":{
          "type":"string",
          "analyzer": "label_analyzer"
        },
        "lang":{
           "type": "string",
           "index": "not_analyzed"
        }
      }
    },
          "uri":{
        "type": "string",
            "index": "not_analyzed"
          },
      "dataSource":{
        "type": "string",
            "index": "not_analyzed"
          }
     }
    }
   }
}'
```

Listing 5.5: ElasticSearch mapping of ontology labels.

As seen in Listing 5.5, an index created with name *sdo* (SDO ontology is used for case study and this ontology is explained in Section 7.2). *sdo* index contains a mapping named as *ont_class* which is used to index ontology classes. Each *ont_class* mapping contains *label*, *uri* and *dataSource* information for classes.

5.1.3 Document data model

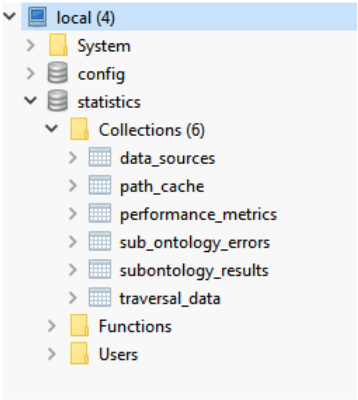
The document data model is another component of the polyglot semantic data model. This data model is used for storing log records, statistics and metadata of data sources. The document data model has been developed using MongoDb document database (Banker et al., 2016). The document data model meets the four different needs of the semantic data model. These requirements and corresponding MongoDb collections are listed and detailed below:

- **Caching:** The most costly part of the ontology modularization method is finding paths between selected concepts. Caching is performed so that paths identified once can be reused without any cost. Caching data is stored in **path_cache** collection.
- **Ontology resource management:** The ontology resource manager handles metadata about ontology resources registered in the semantic data model. This metadata includes the name of the ontology, description, relation with

other registered ontologies, connection string data for graph and full-text data sources. When one needs to access an ontology for query or traversal purposes, they need to get connection string data from ontology resource manager. Ontology metadata is stored in **data_source** collection.

- **Statistic management:** The statistics manager stores statistical information about specific operations performed during the ontology modularization process. This information is used in performance monitoring and improvement studies. Monitored performance data is listed below:
  - *Basic performance metrics:* Basic performance metrics are stored in **performance_metrics** collection. This data contains the information about how long each ontology modularization operation takes.
  - *Sub-ontology results:* Output of all ontology modularization operations is stored in **subontology_results** collection.
  - *Traversal data:* Traversal data (visited nodes, path lengths, etc.) is stored in **traversal_data** collection.
- **Exception management:** Exception manager stores details about occurred exceptions during reuse and modularization process, such as error message, exception stack trace and selected concepts. **sub_ontology_errors** collection is used for keeping all exception information.

All components of the document data model are designed as 6 separate collections on MongoDb as stated above. These collections are shown in the Figure 5.7.



- local (4)
  - System
  - config
  - statistics
    - Collections (6)
      - data_sources
      - path_cache
      - performance_metrics
      - sub_ontology_errors
      - subontology_results
      - traversal_data
    - Functions
    - Users

Figure 5.7: Document model collections.

Each collection represents a nested document model. Therefore, it is important to show details stored in the collections. Therefore, the entire document data model of the collections is expresses in the Figure 5.8

5.2 Semantic Cloud Services

Semantic cloud services run on top of the semantic data model which has been developed with a scalable polyglot architecture. These services have also been developed in a flexible and scalable manner in accordance with the requirements of cloud architecture.

Semantic cloud services are developed by using Spring Boot (SpringBoot, 2020) infrastructure on Java Platform. Although all services currently work in a single process, it is possible to run each of these services in a separate container.

All services are developed in a modular structure. Each module consists of components that perform specific tasks. Thus, the common and shared components can be reused by different modules. It is possible to see these modules and their relations in the module diagram shown in Figure 5.9. Each module is also detailed below[41].

- **ontorogist-service:** This module includes contract definitions and implementations of semantic cloud services. It also contains definitions for REST endpoints.
- **ontorogist-core:** All components related to ontology modularization and path algorithms are included in this module. Access to the semantic cloud data model, search components, statistic managers, log managers, and all other process operations are performed by this module.
- **utils:** It includes helper components that can be used by all modules, such as configuration management, file operations and string operations.
- **ontorogist-shared:** It contains entity definitions that can be used by all modules of the application. Basic classes such as Vertex, Edge, Pathway, Ontology and Ontology Module are defined in this module.

[41]

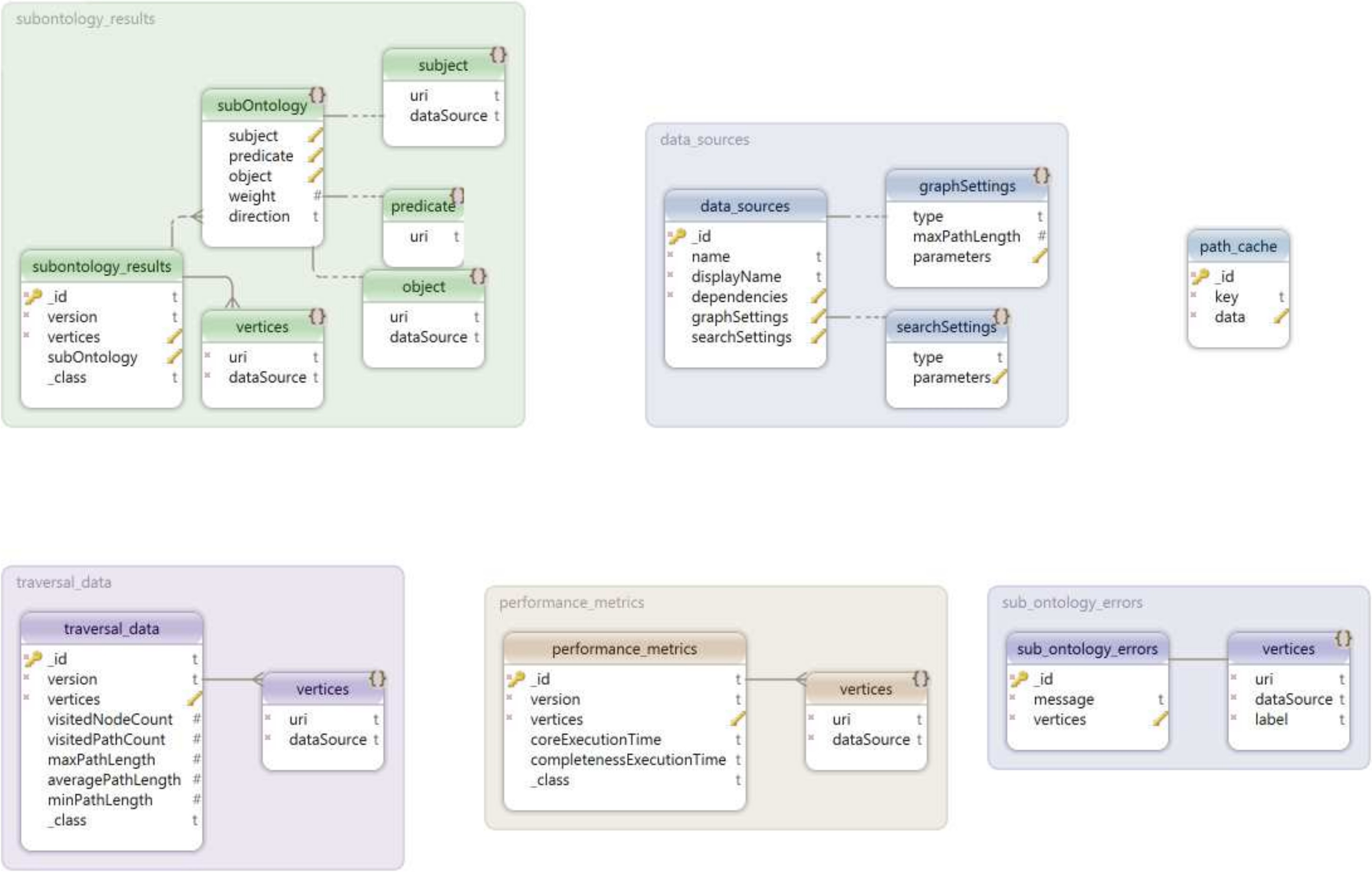


Figure 5.8: Document data model.

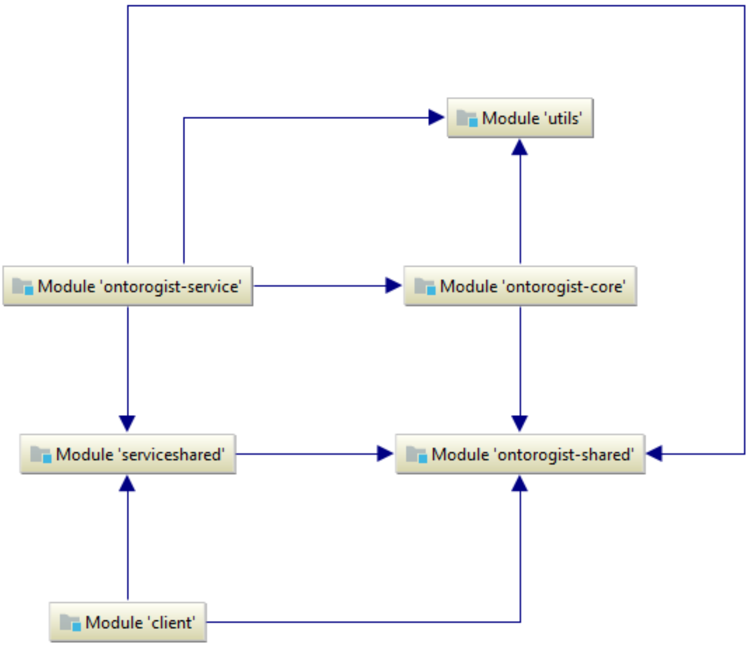


Figure 5.9: Modules of semantic cloud services.

- **serviceshared:** It contains definitions of request and response contracts of semantic cloud services. Used by client modules as well as service modules.
- **client:** Client module contains Protege plugin codes and will be detailed in next section (see Section 6).

The ontology reuse and modularization application is named as *ontoRogist*, and it is inspired by the word ontologist. Ontologist means that an expert or student who deals with ontologies. The letter l in ontologist is replaced with capital R to emphasize reuse. Therefore, there is an *ontorogist* prefix in all package names of the application.

Semantic cloud services have been developed as 3 sub-services using the module components mentioned above: Search service, ontology service and path service.

5.2.1 Search service

The search service is a sub-service of semantic cloud architecture where the ontology data indexed in the full-text data model can be queried. This service currently offers a single method to search only indexed text data (labels and descriptions) of ontology sources. In the future, it is planned to add methods that can be used for further and complex search operations. The current service method is shown in Table 5.5.

Table 5.5: Methods of search service.

Search service method	Description
search	Searches indexed label data of ontologies

The full-text search data model and search service have been developed as an adaptable and customizable infrastructure like other data models. Integration with a different search engine is also possible using base APIs. The class diagram showing the relevant classes and their ElasticSearch implementations used in the development of this data service is shown in Figure 5.10. The classes and interfaces shown in this diagram correspond to the implementation detail of the “Search Components” which are illustrated in Figure 5.1.

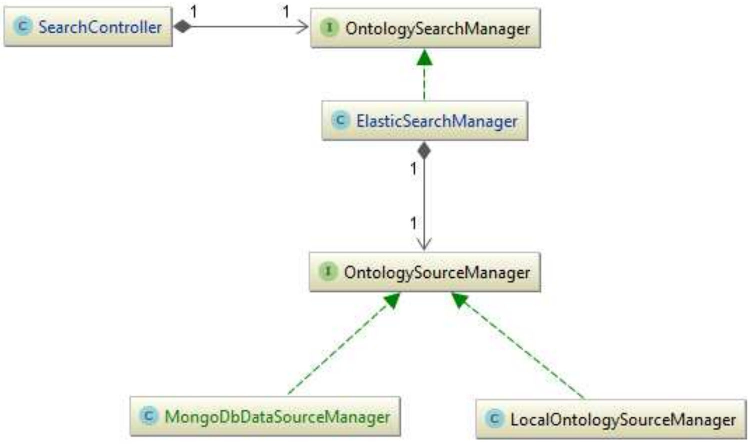


Figure 5.10: Class diagram of search service.

5.2.2 Ontology metadata service

The ontology metadata service is a semantic cloud service where queries can be performed about the members (classes, properties and definitions) of ontologies. It is actually a metadata service, and modularization operations are out of its scope. This service does not include all possible ontology metadata methods in its current version. Only the required methods have been developed and other methods will be implemented in the future.The current methods of ontology metadata service is given in Table 5.6.

Table 5.6: Methods of ontology service.

Ontology metadata service method	**Description**
getClass	Returns the details (label and data source) of a class with given URI identifier. Returned data does not contain relations of given URI.
getClasses	Returns the details of classes with given URI identifiers
getDefinition	Returns all relations of class with given URI
getProperties	Returns all properties of class
getSuperClasses	Returns all superclasses of class
getEquivalentClasses	Returns all equivalent classes of class

The ontology metadata service runs on the Neo4j database. The interfaces and implementations of this service are shown in detail in the class diagram in Figure 5.11. The classes and interfaces shown in this diagram correspond to the implementation detail of the “Metadata Components” which are illustrated in Figure 5.1.

5.2.3 Path service

Path service is a service that provides the methods that carry out the main process of ontology modularization. It handles the processes of finding semantic paths between concepts and ontology modularization. Therefore, it has more complicated structure than other two services. All components of path service runs on Neo4J APIs like a data access layer in which path and search algorithms are operated. The components and details of this access layer are illustrated in Figure

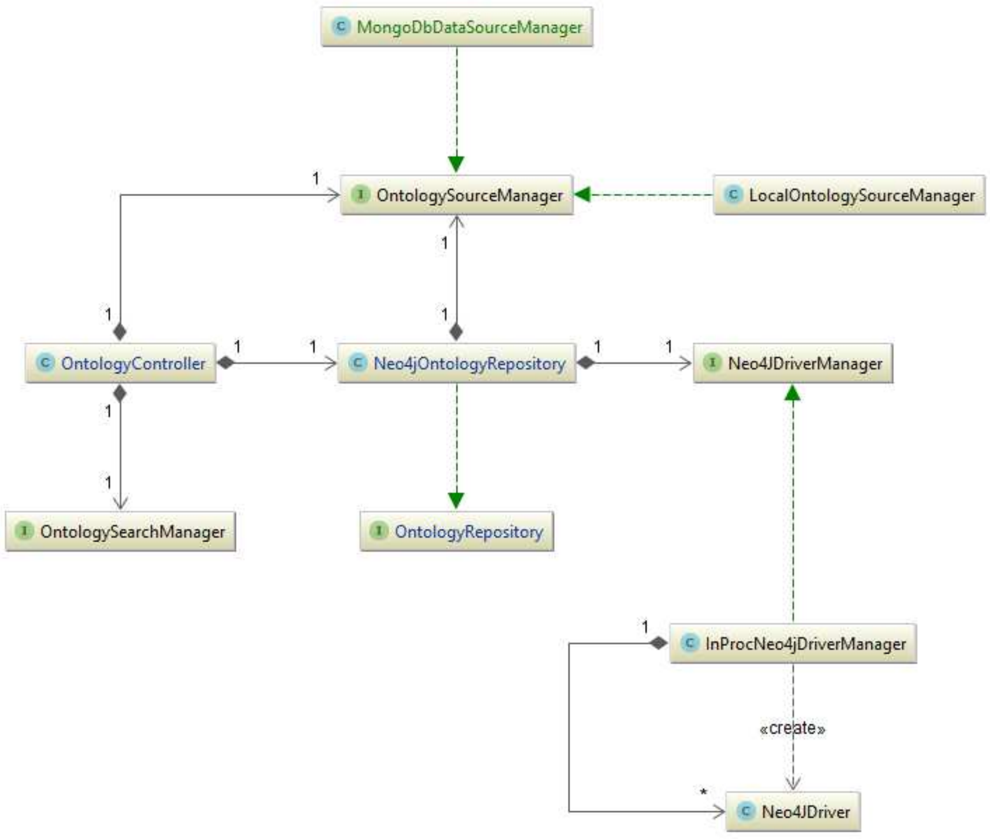


Figure 5.11: Class diagram of ontology metadata service components.

5.12 and detailed below:

- **Search coordinator:** It manages and controls BFS traversals that run bidirectionally and simultaneously from two sides.
- **Relatedness calculator:** It calculates semantic relatedness values of paths, by using OWL relatedness coefficients recommended by Giray and Unalir (Giray and Ünalır, 2013) as default. It is possible to customize and adapt relatedness calculations by using different methods.
- **BFS:** It starts the BFS algorithm from one side and navigates using path filters.
- **Path filters:** They filter path traversals by using the semantically correct path (Hirst and St-Onge, 1998) restrictions and path length parameters.
- **Neo4J API:** They are native APIs provided by Neo4j.

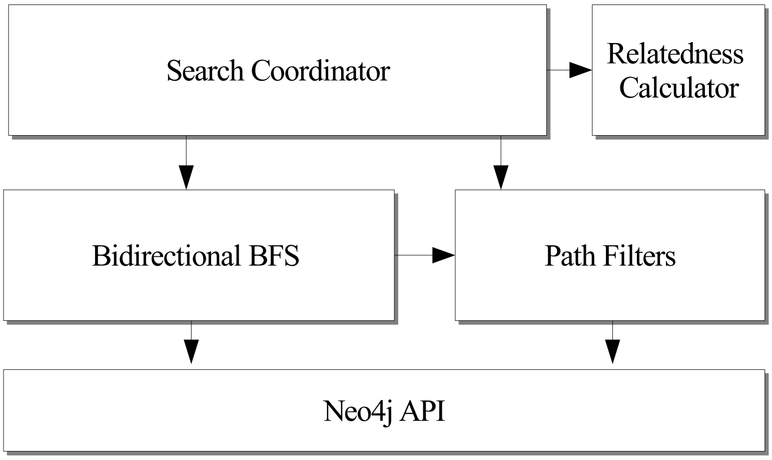


Figure 5.12: Data access layer of graph data model.

In the current version, the path service performs path finding and modularization. The methods that carry out these procedures are listed in Table 5.7.

Table 5.7: Methods of path service.

Path service method	Description
findPath	Finds and returns most semantic path between given concepts
findSubOntology	Builds and returns sub-ontology as triple list for given concept list
downloadSubOntology	Builds and returns sub-ontology as a valid OWL file for given concept list

Since the number of components (classes) used by this service is high, only specific dependencies are shown on the diagram in Figure 5.13 for simplicity. The classes and interfaces shown in this diagram correspond to the implementation detail of the “Path and Module Extraction Components” which are illustrated in Figure 5.1.

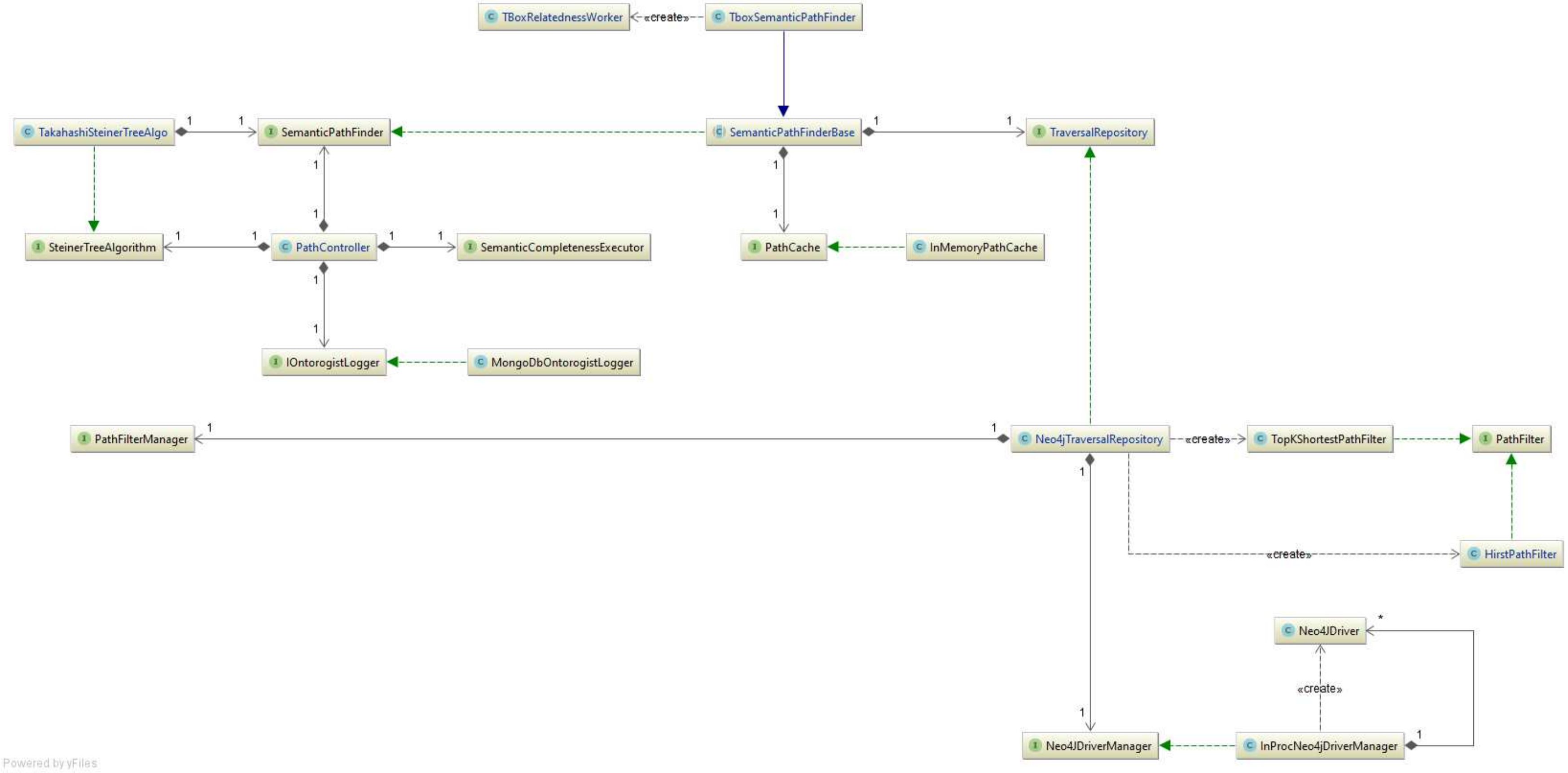


Figure 5.13: Class diagram of path service components.

5.3 Extract-Transform-Load Tool

Due to the requirements within the scope of this study, a polyglot semantic data model has been developed in order to store ontologies. However, since this data model is not an ontology database, ontologies cannot be loaded directly. As a result, an Extract-Transform-Load (ETL) tool was needed to make this transfer.

Ontologies are stored in graph and full-text data models of the semantic data model. Therefore, the ETL tool must transfer data to these two data models using their internal representations. The document data model is not included in the transfer process, as it stores metadata, cache content, and log records about ontologies.

The ETL tool consists of 4 sub-components. These components and their relations with each other are shown in Figure 5.14.

The internals of each component of the ETL tool are detailed below:

- **N-Triple Converter:** It converts ontology files into N-Triple format if it is defined with a different format. Thus, it is possible to process the file line by line and in parallel.
- **Content Cleaner:** In fact, this component works inside bulk converter components (Neo4j bulk converter and Elasticsearch bulk converter). It filters the definitions that are not required to be transferred to the semantic data model.
- **Neo4j Bulk Converter**: It is a very costly and long process to transfer all triples to the Neo4j database by reading them line by line. Instead, the bulk import feature of the Neo4j database is used. The triple definitions are converted to Neo4j's internal CSV format.
- **ElasticSearch Bulk Converter:** Bulk capabilities of the ElasticSearch database are used, similar to the bulk import feature used to transfer data to the Neo4j database. The triple data is converted to the JSON format of the ElasticSearch database by bulk converter tool.
- **Neo4j Loader:** This component transfers the ontology converted to CSV format to the Neo4j database using bulk import.

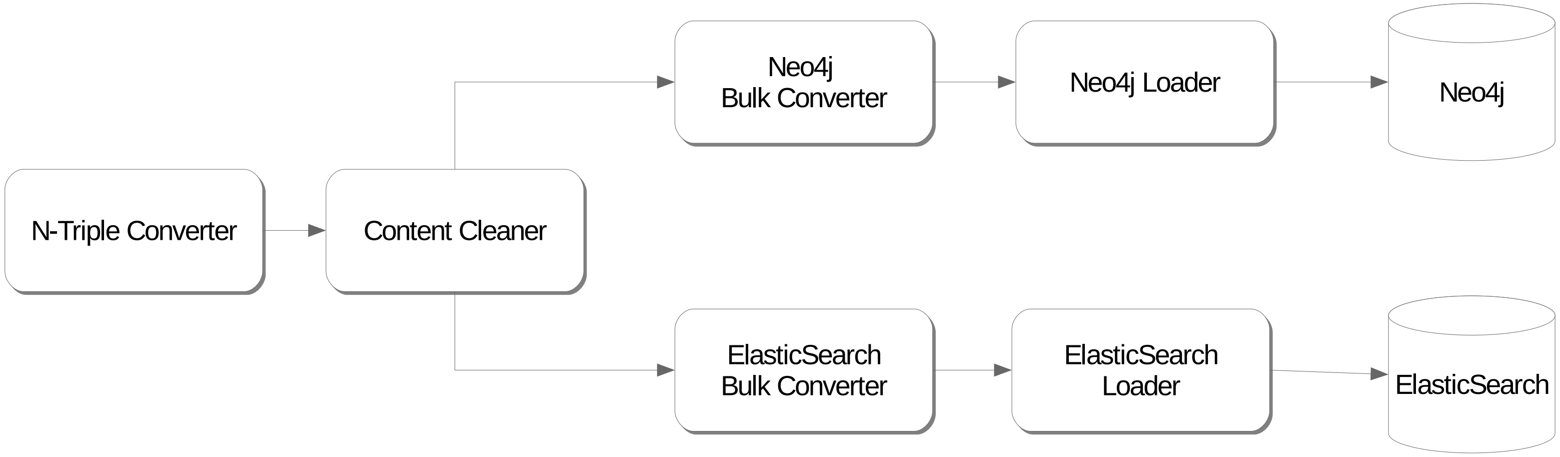


Figure 5.14: Components of ETL tool.

- **ElasticSearch Loader:** This component transfers the annotation and label data of ontology, which is previously converted to JSON format, to the ElasticSearch database using bulk import.

5.4 Conclusion

The aim of this study is not only to propose a method that supports reuse, but also to make it a usable final product. Therefore, the reuse method was developed with a elastic and scalable data model and semantic cloud services running on this data model. Adaptability has always been considered in the development of these services and the data model. In this way, it is aimed to easily add the changes that can be made at any time.

6. DEVELOPMENT OF THE TOOL TO SUPPORT ONTOLOGY REUSE

One of the major obstacles to ontology reuse is the lack of a solution where ontology reuse and modularization methods are integrated with ontology reuse tools. Extensions and interfaces to support ontology engineers for both methodology, process and method will significantly increase usability. Thus, it is thought that the performance of the method can be increased further.

When the ontology modularization methods in the literature are examined, it is seen that the methods supporting plugins and user interfaces are quite few. For example; (Ranwez et al., 2012) provides a command line based access for their modularization method. Although Noy and Musen (2004) provides a Protege extension with their modularization method, this extension is outdated and unavailable. Both MIREOT and Bioportal Import Plugin offer Protege plugins but they do not actually support ontology modularization.

The aim of this study is not only to propose an ontology reuse methodology and method, but also to make this method usable with various user interfaces. These user interfaces will simplify the development process for ontology engineers and increase the usability and performance of the method.

This section describes the interfaces and service endpoints that provide access to the ontology reuse method. All service components are exported with REST endpoints. These endpoints allow components to be integrated with external applications. In addition, two client applications have been developed in which the ontology reuse method can be used. The first of these applications is in the form of a web portal. The other is a Protege plugin. Both applications perform all processes via REST endpoints.

6.1 Scenarios of ontology reuse

The use case scenarios identified during the development of the application are shown in Figure 6.1. In all scenarios, the ontology engineer is the only actor.

The sub-steps of use case scenarios are detailed below:

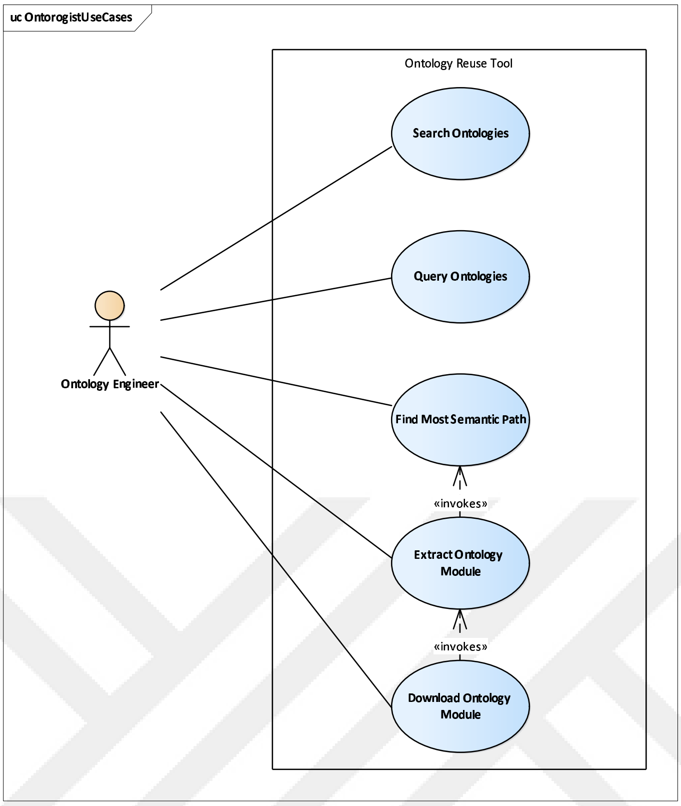


Figure 6.1: Use case diagram for ontology reuse tool.

- **SearchOntologies:** Ontology engineers should be able to perform full-text searches on registered ontologies using keywords.
  - User should be able to determine which ontology resources they want to search.
  - Text to be searched should be entered in accordance with full-text search format.
  - Returning results of each ontology should be discriminated.
  - If the search result is empty, the user should be notified.
  - The user should be able to select the concepts returned by the search result and include them in the ontology.
- **QueryOntologies:** Ontology engineers should be able to perform detailed queries about concepts included in ontology. It is possible to summarize these queries as below:
  - All relations
  - Super-classes

  - Equivalent classes
  - Properties
  - Domain/range definitions

- **FindMostSemanticPath:** The ontology engineer should be able to find the most semantic path between the two concepts. The steps and requirements for this usage scenario are listed below:

  - User should be able to find all path alternatives between two concepts.
  - The most semantic path between the two concepts should be identified.
  - If there are more than one path between the two concepts, the user should be able to select the paths other than most semantic one.

- **ExtractOntologyModule:** The ontology module extraction scenario uses the ontology path finding method directly. The steps for this scenario are listed below:

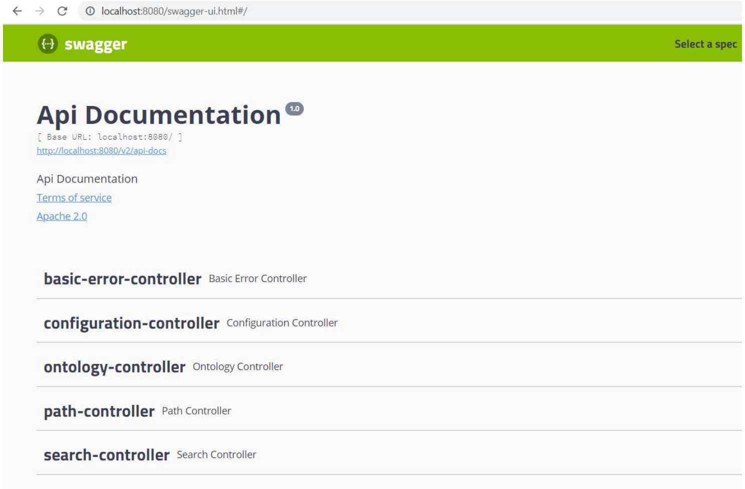
  - User should be able to create an ontology module by selecting more than two concepts.
  - The size of the module created must be minimum.
  - User should be able to expand module using parameters.
  - An existing module should be integrated with concepts or paths to form a new module.

- **DownloadOntologyModule:** Ontology engineers should be able to download extracted ontology module as an OWL file. All other requirement of this scenario is same with ExtractOntologyModule**.**

6.2 REST Endpoints of Semantic Cloud Services

In Section 5, service methods exported as REST endpoints has been mentioned. This section will list the request and response parameters of related REST methods in detail. It is possible to access the details of all REST methods via the swagger[42] endpoint. This endpoint is hosted at http://localhost:8080/swagger-ui.html (See Figure 6.2).

[42] https://swagger.io/



Api Documentation 1.0

[Base URL: localhost:8080/]

http://localhost:8080/v2/api-docs

Api Documentation

Terms of service

Apache 2.0

basic-error-controller Basic Error Controller

configuration-controller Configuration Controller

ontology-controller Ontology Controller

path-controller Path Controller

search-controller Search Controller

Figure 6.2: Swagger endpoint.

Some of the methods are not listed here for simplicity, only important methods are included. Service methods and their parameters are shown on tables Table 5.5, Table 6.2 and Table 6.3.

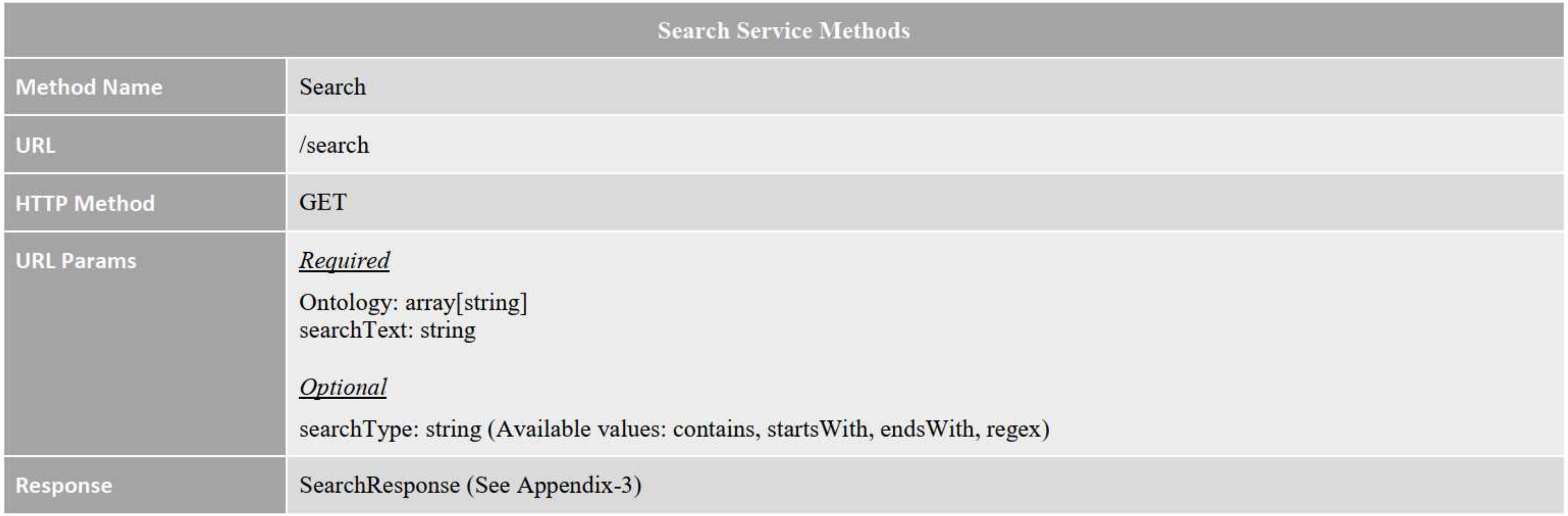
6.3 Ontology Reuse Web Application

The web application is an end-user application developed using HTML, Javascript and REST endpoints served by ontology reuse server. The operations that can be performed on this application are listed below:

- Concepts can be searched by keywords.
- Sample data sets are available on the page.
- Contract information of REST endpoints are accessible.
- Ontology module can be extracted.
- Extracted ontology modules can be downloaded as OWL file.

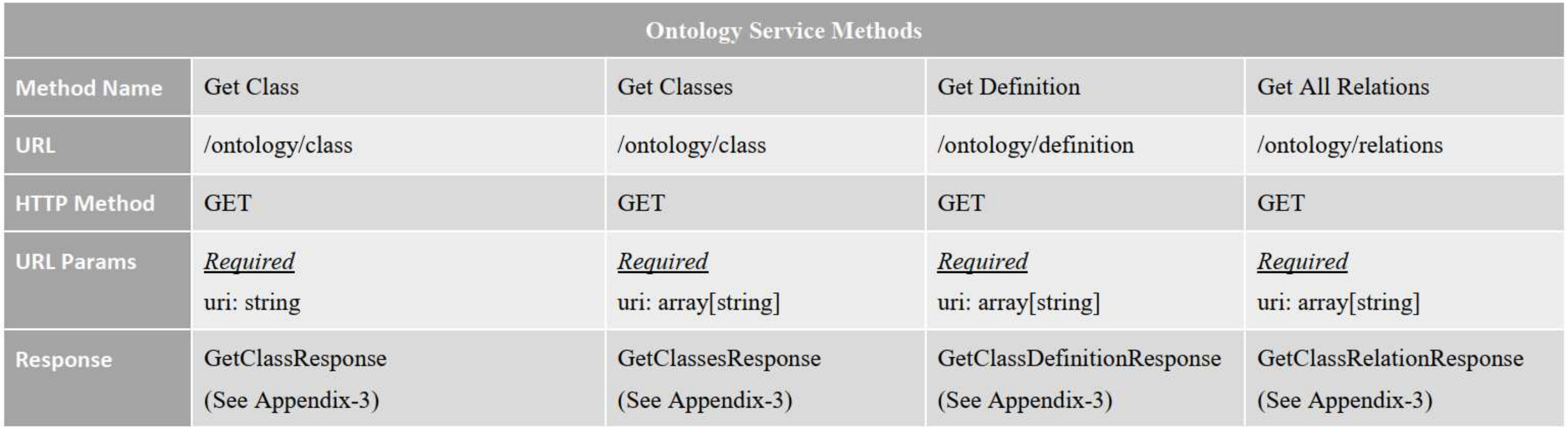
A sample screenshot of web application is shown in Figure 6.3.

Table 6.1: Search service methods.



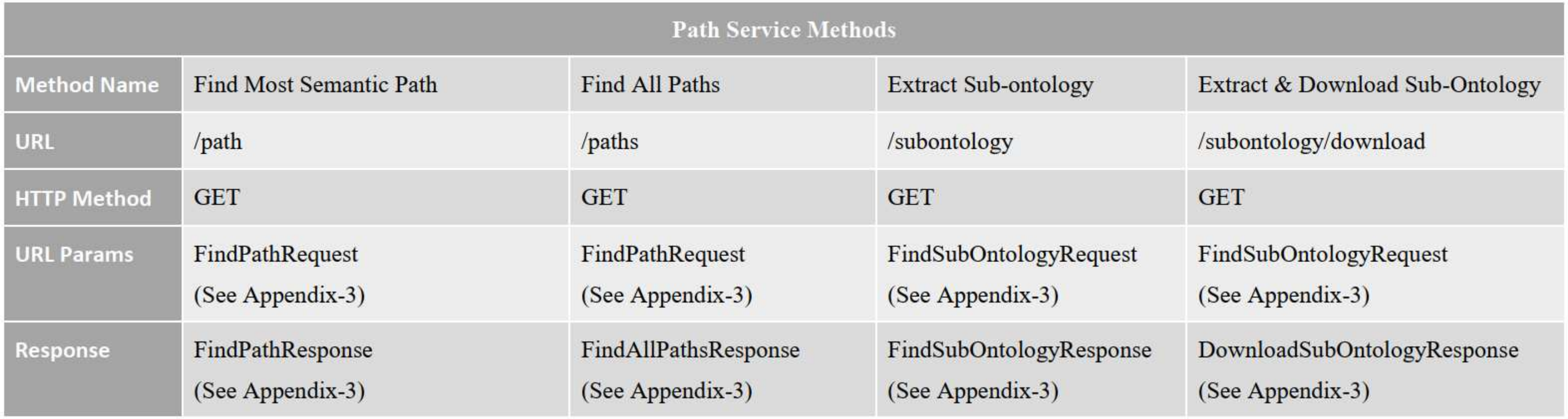
Search Service Methods	
Method Name	Search
URL	/search
HTTP Method	GET
URL Params	*Required* Ontology: array[string] searchText: string *Optional* searchType: string (Available values: contains, startsWith, endsWith, regex)
Response	SearchResponse (See Appendix-3)

Table 6.2: Ontology service methods.

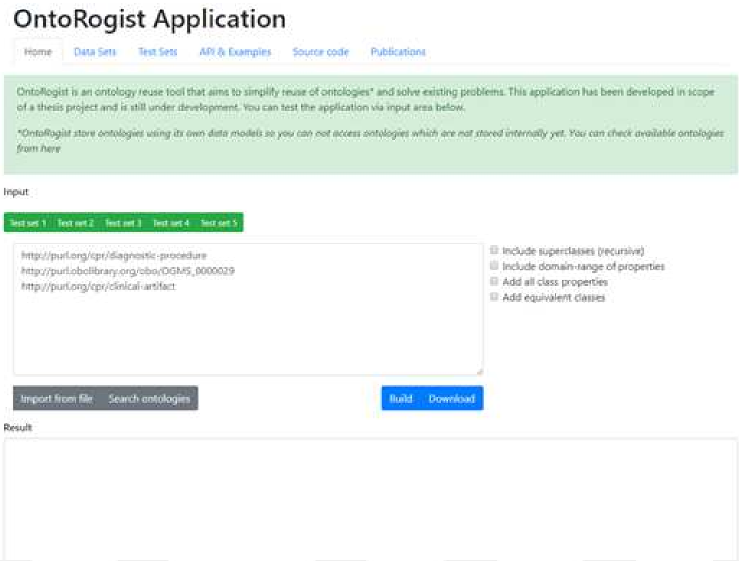


Ontology Service Methods				
Method Name	Get Class	Get Classes	Get Definition	Get All Relations
URL	/ontology/class	/ontology/class	/ontology/definition	/ontology/relations
HTTP Method	GET	GET	GET	GET
URL Params	*Required* uri: string	*Required* uri: array[string]	*Required* uri: array[string]	*Required* uri: array[string]
Response	GetClassResponse (See Appendix-3)	GetClassesResponse (See Appendix-3)	GetClassDefinitionResponse (See Appendix-3)	GetClassRelationResponse (See Appendix-3)

Table 6.3: Path service methods.



Path Service Methods				
Method Name	Find Most Semantic Path	Find All Paths	Extract Sub-ontology	Extract & Download Sub-Ontology
URL	/path	/paths	/subontology	/subontology/download
HTTP Method	GET	GET	GET	GET
URL Params	FindPathRequest (See Appendix-3)	FindPathRequest (See Appendix-3)	FindSubOntologyRequest (See Appendix-3)	FindSubOntologyRequest (See Appendix-3)
Response	FindPathResponse (See Appendix-3)	FindAllPathsResponse (See Appendix-3)	FindSubOntologyResponse (See Appendix-3)	DownloadSubOntologyResponse (See Appendix-3)



OntoRogist is an ontology reuse tool that aims to simplify reuse of ontologies* and solve existing problems. This application has been developed in scope of a thesis project and is still under development. You can test the application via input area below.

OntoRogist store ontologies using its own data models so you can not access ontologies which are not stored internally yet. You can check available ontologies from here

http://purl.org/cpr/diagnostic-procedure
http://purl.obolibrary.org/obo/OGMS_0000029
http://purl.org/cpr/clinical-artifact

□ Include superclasses (recursive)
□ Include domain-range of properties
□ Add all class properties
□ Add equivalent classes

Figure 6.3: Screenshot of web application.

6.4 Protege Plugin

Protege (Musen, 2015) is a modular application and has an extensible architecture. OSGI[43] framework is used in Protege as plugin infrastructure. By making use of this structure of Protege, ontology reuse tool has been developed as a Protege plugin. It is thought that it would be more beneficial to develop a plugin for Protege, which has become the de-facto standard for ontology development, instead of developing a completely independent tool.

The ontology reuse plugin developed supports many of the functions offered through REST services. However, the enrichment of the plugin in terms of functionality is also planned. The functions the plugin currently supports are listed below:

- Visualization
- Searching
- Path finding
- Ontology-modularization

[43] https://www.osgi.org/

The reuse plugin has been developed using the open source OntoGraf[44] Protege plugin. The Protege reuse plugin must first be activated through the Protege menu (See 6.4). As mentioned in previous section (see Section 5), application is named as *ontorogist.* Same name is also used for Protege plugin name.

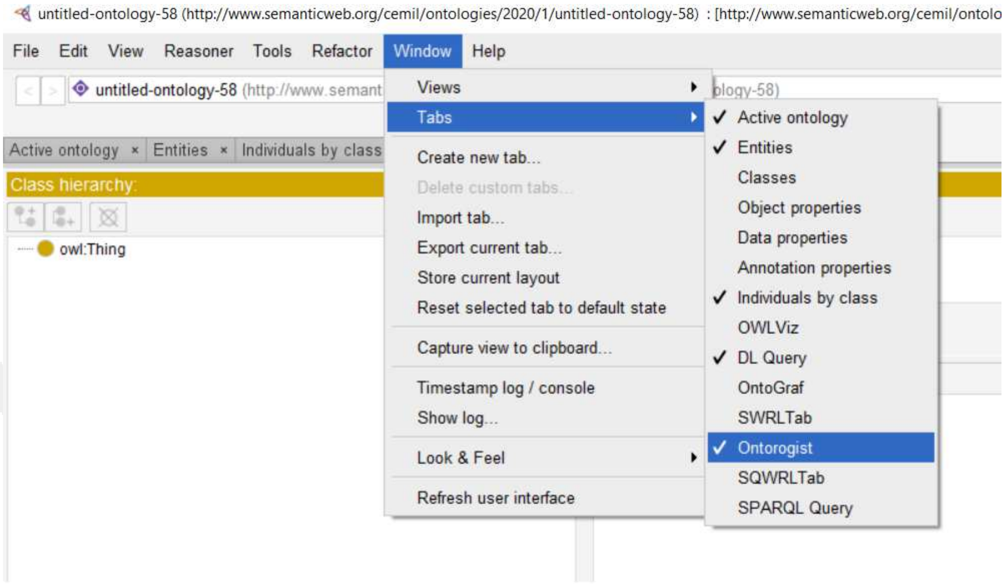


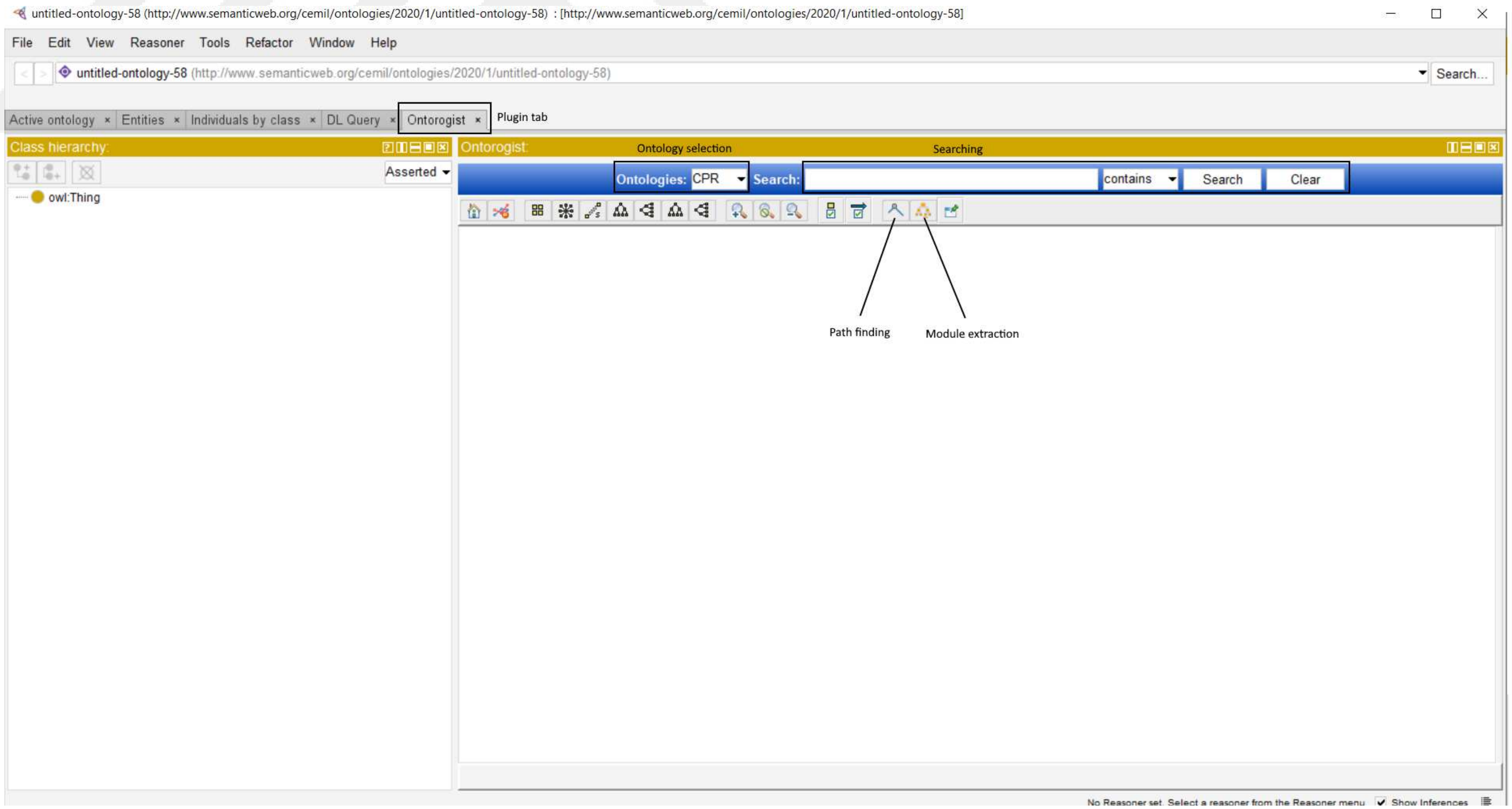
Figure 6.4: Activating reuse plugin.

The plugin appears as a tab in the Protege application. The appearance and basic functions of this tab are shown in Figure 6.5.

6.4.1 Searching

Users can search by entering text on the search box on the main window. Search results will be displayed on a separate dialog (see Figure 6.6). The concepts chosen here will be transferred to the design interface of the main screen (see Figure 6.7). New searches can be made through the dialog where the search results are displayed.

[44] https://github.com/protegeproject/ontograf


untitled-ontology-58 (http://www.semanticweb.org/cemil/ontologies/2020/1/untitled-ontology-58) : [http://www.semanticweb.org/cemil/ontologies/2020/1/untitled-ontology-58]
File Edit View Reasoner Tools Refactor Window Help
untitled-ontology-58 (http://www.semanticweb.org/cemil/ontologies/2020/1/untitled-ontology-58)
Search...
Active ontology ×
Entities ×
Individuals by class ×
DL Query ×
Ontorogist ×
Class hierarchy:
Asserted
owl:Thing
Ontorogist:
Ontologies: CPR
Search:
contains
Search
Clear
No Reasoner set. Select a reasoner from the Reasoner menu
Show Inferences

Plugin tab

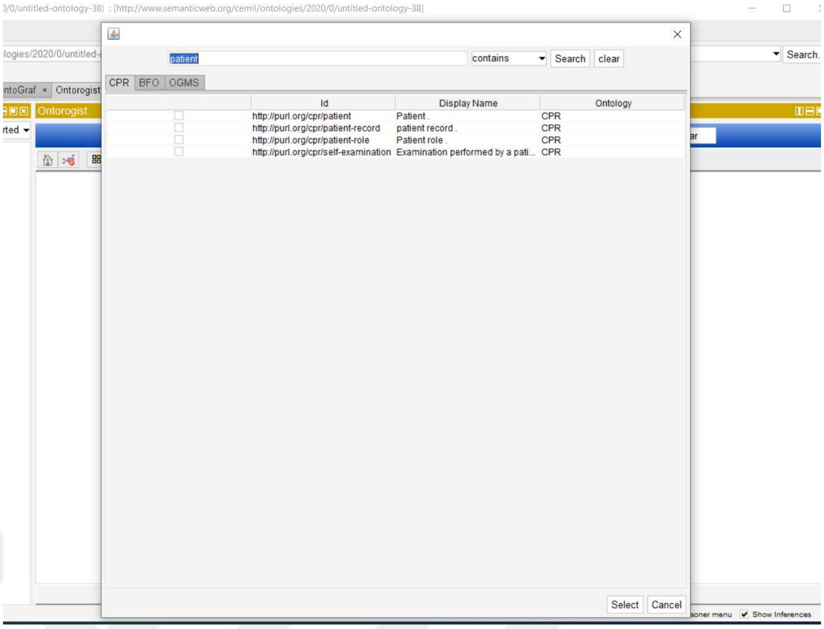
Ontology selection

Searching

Path finding

Module extraction

Figure 6.5: Plugin main window.

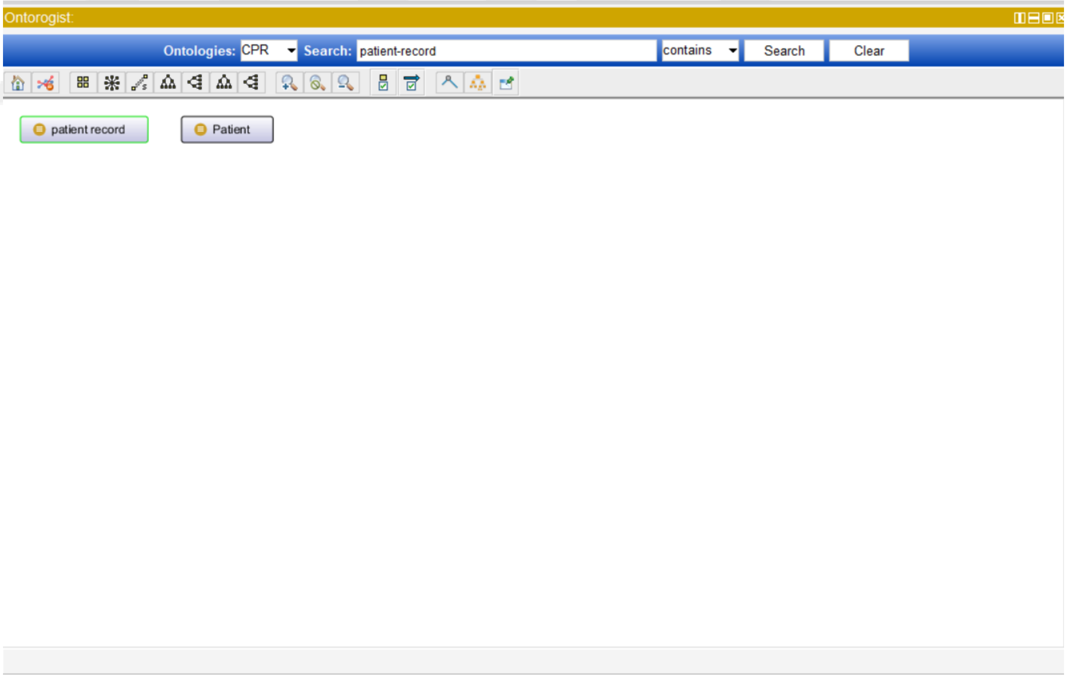


patient | contains | Search | clear

	Id	Display Name	Ontology
□	http://purl.org/cpr/patient	Patient .	CPR
□	http://purl.org/cpr/patient-record	patient record .	CPR
□	http://purl.org/cpr/patient-role	Patient role .	CPR
□	http://purl.org/cpr/self-examination	Examination performed by a pati...	CPR

[http://www.semanticweb.org/cemil/ontologies/2020/0/untitled-ontology-38]
CPR
BFO
OGMS
Select
Cancel
Show Inferences

Figure 6.6: Search results dialog.

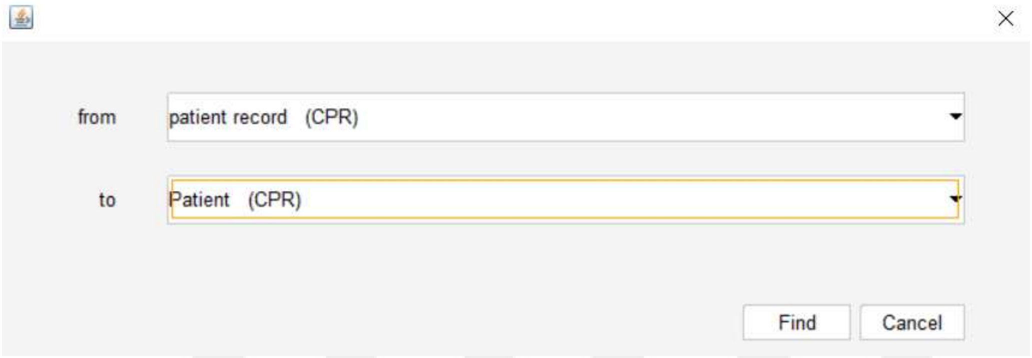


Ontologies: CPR | Search: patient-record | contains | Search | Clear

Figure 6.7: Selecting from search results and adding main designer window.

6.4.2 Path finding

The concepts added on the main design view can be selected in pairs to find paths between them. A dialog box will be opened for the selection of the concept pair for this process (see Figure 6.8). If more than one path is found, possible paths will be listed on a selection window (see Figure 6.9). If the user chooses a path through this selection window, the relevant path will be added to the design view (see 6.10).



from patient record (CPR)

to Patient (CPR)

Figure 6.8: Concept selection dialog for path finding.

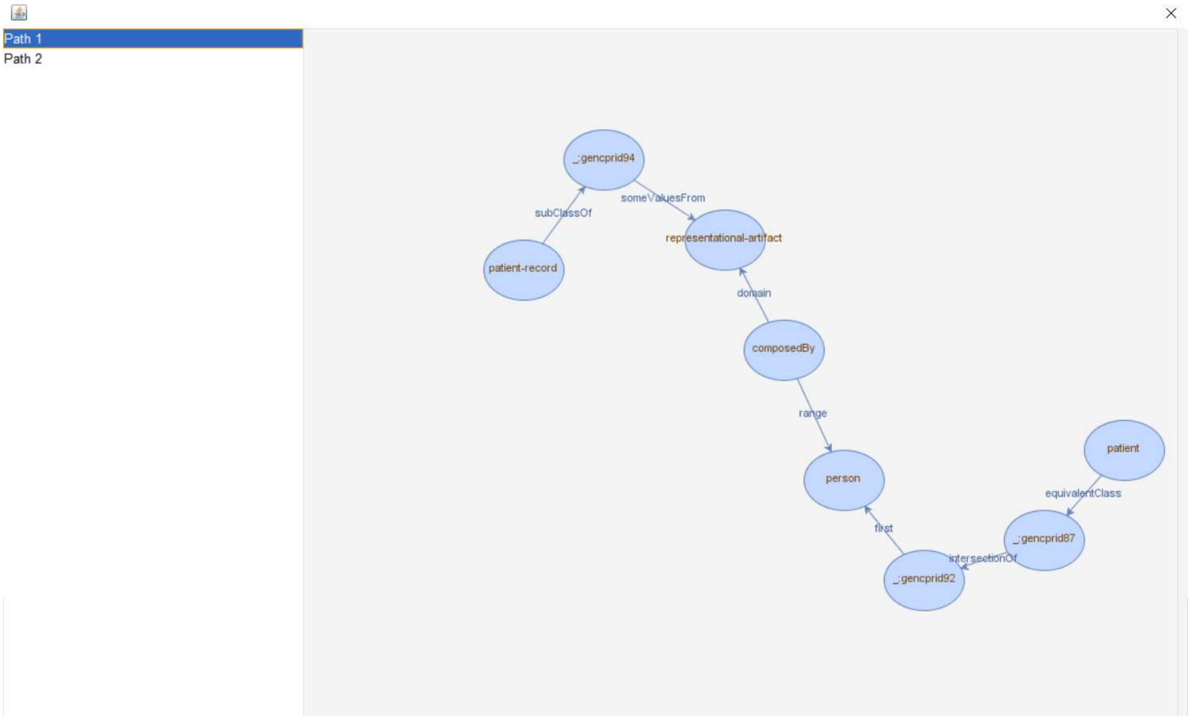
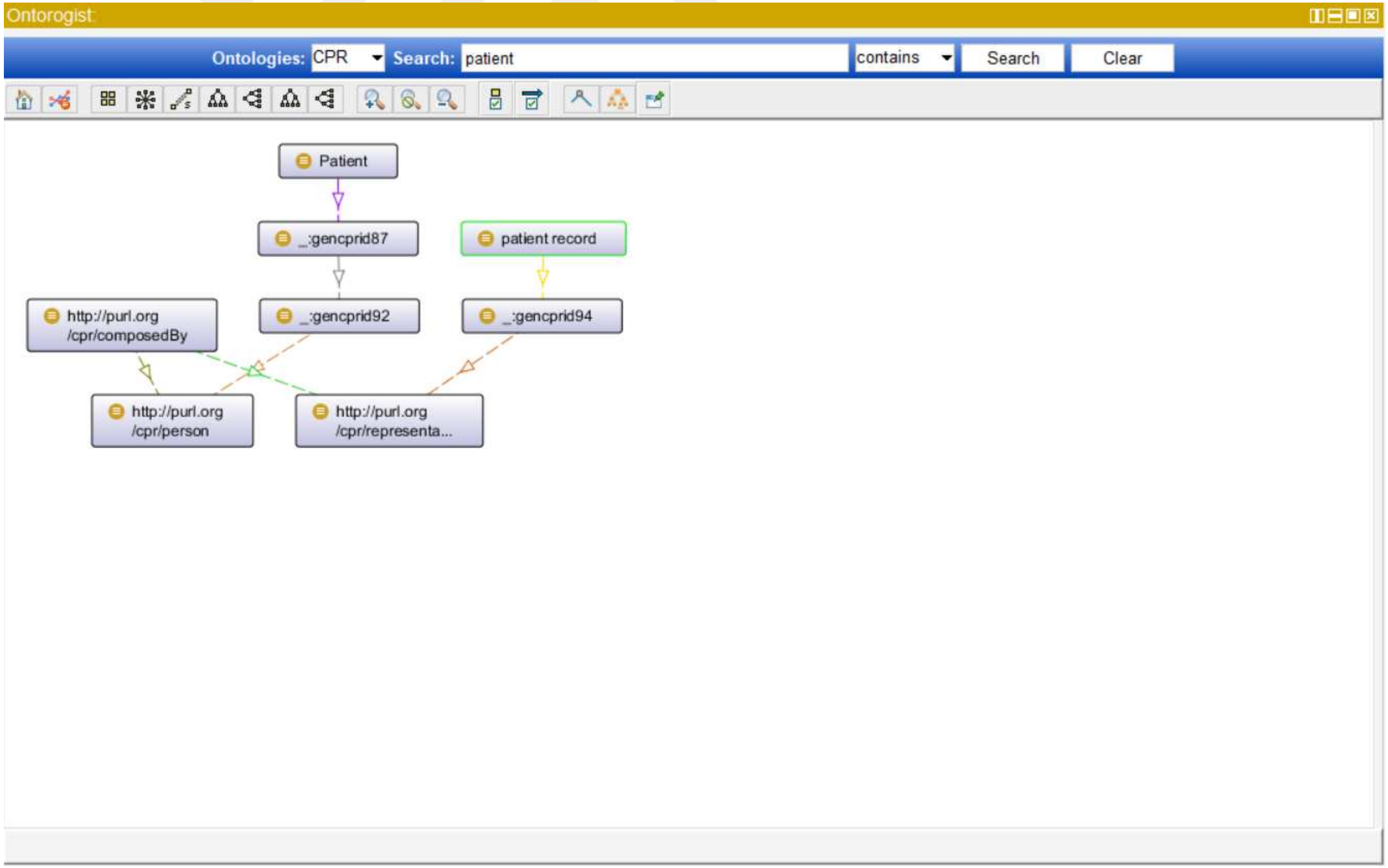


Figure 6.9: Path result dialog.



Ontorogist

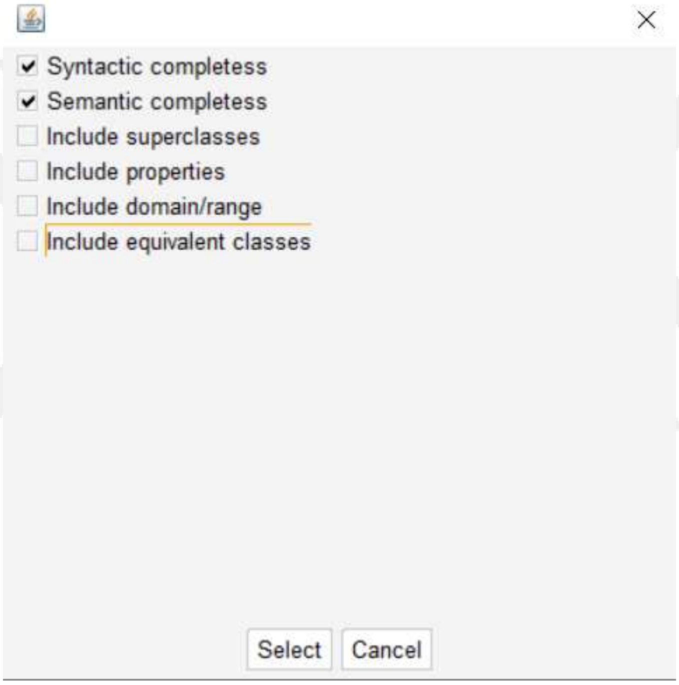
Ontologies: CPR Search: patient contains

Search
Clear
Patient
_:gencprid87
patient record
http://purl.org /cpr/composedBy
_:gencprid92
_:gencprid94
http://purl.org /cpr/person
http://purl.org /cpr/representa...

Figure 6.10: Adding selected path to the main view.

6.4.3 Ontology module extraction

The ontology module extraction process creates the ontology module where all the concepts on the main design screen are linked. Unlike the path selection, the concept selection is not made by the users. Syntactic and semantic completeness parameters are asked to the user through a window and ontology module extraction is performed using these parameters (see Figure 6.11). The result of ontology module extraction is added to the main design view.



☑ Syntactic completess
☑ Semantic completess
□ Include superclasses
□ Include properties
□ Include domain/range
□ Include equivalent classes

Figure 6.11: Ontology module extraction parameters.

6.5 Conclusion

In this section, information about tools to support ontology reuse and details of these tools are given. Not only REST endpoints of application developed , but also a web application and a Protege plugin. This application and plugin is important in terms of increasing the usability of the developed method and clarifying its aims.

7. A CASE STUDY: SLEEP DOMAIN ONTOLOGY

In this section, a case study conducted within the proposed method will be mentioned. Criteria and results used to evaluate performance will also be included.

7.1 How to Evaluate Ontology Reuse?

Evaluation studies on ontology reuse are generally concerned with the reuse rates of existing ontologies. The recent studies in Kamdar et al. (2016), Ochs et al. (2017) and Vandenbussche et al. (2017) are prominent examples within this context. However, there is no method and study on how to evaluate the ontology reuse process. The lack of well-accepted guidelines and methods for ontology reuse is the biggest obstacle for the evaluation of the reuse process.

Ontology reuse is a comprehensive and semi-automated process involving many sub-steps such as search, evaluation, ontology module extraction and integration. Since the ontology reuse process should be semi-automated, interaction with ontology engineers is often needed. However, it is not possible to measure or evaluate user-interactive sub-steps. Therefore, validation studies should be performed only on the automated sub-process, relatedness-based ontology modularization method.

As mentioned in section 2.5, there are many ontology modularization methods in the literature. However, there is no accepted evaluation criteria. Each method carries out an evaluation study in accordance with its own point of view. This problem is also clearly stated by d'Aquin et al. (2009) and Doran et al. (2007). It was emphasized that the inability to define what a good ontology module is, prevents finding out a well-defined evaluation criteria. Table 7.1 summarizes which evaluation criteria is used by the existing ontology modularization methods. Understanding these criteria is important for determining how to evaluate the method developed within the scope of this thesis.

As seen in Table 7.1, performance measurements are mainly performed within the scope of the assessment. In these performance measurements, the methods are not compared between each other and it is only emphasized how fast the current method can work.

Table 7.1: Evaluation criteria used by existing modularization methods

Method	**Evaluation criteria**
(Noy and Musen, 2004)	N/A
(Seidenberg, 2009)	Performance & module size
MOVE (Bhatt et al., 2006)	Performance
(Doran et al., 2007)	F-measure
(Miao et al., 2008)	N/A
SOMET (Doran et al., 2008)	Performance
(Hussain and Abidi, 2009)	Module size
(Ranwez et al., 2012)	Performance

Some methods use the size of the ontology modules as an evaluation criteria. However, this evaluation is not sufficient to determine whether the module being extracted meets user expectations. One of the evaluation studies within the scope of the module size belongs to Doran and his colleagues (Doran et al., 2007). In order to evaluate the module, they used the f-measure calculation, which is the harmonic mean of recall and precision values. Recall and precision calculations are frequently used to evaluate the accuracy of results in information retrieval. Doran et al. (2007) adapted this calculation to the ontology module extraction process. With this adaptation, recall and precision are defined as follows:

- **Recall:** Every definition which is in main ontology is in the extracted module.
- **Precision:** All the parent/child relations which are in the extracted module are also in the main ontology.

Original ontology (O) and candidate module (M) are shown in Figure 7.1 two make recall and precision clear. This evaluation method assumes all data included by module is true positives, and remaining data is false negatives. Since the extracted ontology module will be covered by the original ontology in any case, false positives are not likely to occur. Therefore, the precision value is always 1. However, the problem arises in the recall calculation. All definitions and relationships not included in the module are false negative. Therefore, the larger the module, the larger the recall value.

Doran et al.(2007) extracts ontology modules by selecting a single concept and adding its all relations to ontology module recursively (see Section 2.5.4).

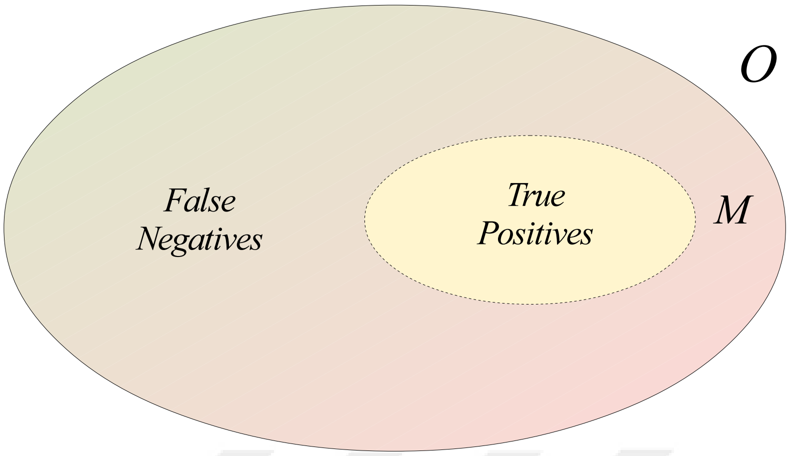


Figure 7.1: Recall/precision calculation proposed by Doran et al. (2007).

Therefore, this evaluation approach can return high recall values in relatively smaller ontologies. It is also possible to reach high values in huge ontologies, but firstly, the applicability and performance of the method in such ontologies should be discussed. A method that recursively tries to include all relationships can cause serious performance problems. Also this method does not works with multiple concept selection like other modularization method.

Let us give an example to detail the problem about recall calculation. Suppose that, a user extract an ontology module from CPR ontology using any other method from Doran et al.(2007), manually or using a ontology modularization algorithm, as shown in Listing 7.1. Also, let's assume that we calculate the recall value with the number of concepts in the ontology module. Doran et al.(2007) uses definitions to calculate recall. In this example, only number of classes is used to simplify calculation. (The ontology module specified in Listing 7.1 is not the output of the method defined in thesis study or another method in literature. It's an example ontology part to calculate recall value).

```
<owl:Class rdf:about="cpr:diagnostic-image">
    <rdfs:subClassOf rdf:resource="cpr:image"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="cpr:outputOf"/>
            <owl:someValuesFrom
             →  rdf:resource="cpr:clinical-investigation-act"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="cpr:diagnostic-procedure">
    <rdfs:subClassOf
     ↪  rdf:resource="cpr:clinical-investigation-act"/>
</owl:Class>
```

Listing 7.1: An ontology module sample to evaluate recall.

In Listing 7.1, there are 4 concept reference, *diagnostic-image*, *image*, *clinical-investigation-act*, *diagnostic-procedure*. So, the number of true positives (TP) is also 4. According to Bioportal statistics, there are 128 classes in CPR ontology[45]. Therefore, for our sample, the number of false negatives (FN) is 124. While recall can be calculated using formula $TP/(TP + FN)$, recall value is 4/(4+124) =0.03125. Although this ontology module meets all the requirements of the user, it is an insufficient module according to the recall value.

In this thesis, recall/precision calculation will not be used as stated by Doran et al.(2007). This evaluation method will be improved by making some customizations.

Taking into consideration the evaluation criteria mentioned above, the module extraction method within the scope of the thesis has been evaluated with 2 different criteria:

- **Performance:** The ontology module extraction process should be able to work with high performance. This evaluation criterion is important for usability.
- **Reuse success:** Does the extracted ontology module meet the reuse needs of users?

The evaluation of these two criteria is explained in the following sections.

[45] https://bioportal.bioontology.org/ontologies/CPRO

7.1.1 Performance evaluation

Performance of ontology module extraction is important for the usability of whole ontology reuse process. Therefore, performance of module extraction method has been evaluated using several criteria:

- Number of selected concepts-module extraction time
- Number of selected concepts-module size
- Number of selected concepts-number of traversed concepts
- Number of selected concepts-number of traversed paths

Each evaluation criteria has been repeated according to conditions such as cache usage and thread pool size. Alongside with these conditions, there are also external factors that directly affects the performance of method, such as size of reused ontology, branching factor and semantic richness of original ontology. Therefore, it is not expected that there is a clear correlation between the selected number of concepts and values such as duration, module size, number of traversed concepts. Some parameters that directly affect the module extraction process are listed below to understand performance evaluation better:

- Size of original ontology
- Average relation count of original ontology (Average branching factor)
- Semantic richness of original ontology (While some ontologies use only taxonomic relationships, some ontologies involve other OWL relations such as restrictions, intersections and unions)
- Position of selected concepts (Selection of concepts that are not close to each other in ontology will increase the module extraction time)

The purpose of the performance evaluation in this study is to verify that the ontology module extraction process can be performed in an acceptable time interval. Since each ontology modularization method defines the requirements of the ontology module to be extracted in different ways, one-to-one comparisons with the existing studies in the literature has not been evaluated.

7.1.2 Recall/precision of ontology reuse

Doran et al. has used recall/precision calculations to evaluate the ontology modularization process (Doran et al., 2007). But, it is not possible to use these calculations to evaluate ontology reuse, and some modifications need to be performed. Doran et al. make these calculations by comparing extracted module with the original ontology. However, a reference point is needed to evaluate reuse. This reference point can be a reliable ontology which has already been developed using existing ontologies. The aim is to develop the reference ontology using the proposed method and tool. While it is not possible directly create reference ontology using reuse tool, at each step, the extracted modules should be evaluated by comparing with reference ontology.

While ontology development and reuse process should be performed iteratively and incrementally, it's not possible to evaluate an existing ontology as a whole by using the ontology modularization method and ontology reuse tool implemented in this thesis study. Instead, a partial approach should be used. Ontology modules should be extracted using selected concepts, and the extracted module should be compared with its counterpart in reference ontology. This approach will be explained step by step. First of all, the formal steps of proposed evaluation method should be listed:

1. Reference ontology (O) should be decomposed using reverse engineering activities. No complex reverse engineering activities are required in this step. Activities that can identify reused ontologies are sufficient.

2. After reverse engineering, each ontology reused by reference ontology ($R_1, R_2, R_3, ...$) should be identified. It is relatively easier to detect imported ontologies, reused with *owl:import*. However, if only URI references are used from existing ontologies, it is required to check all class definitions, and find reused ontologies.

3. All of the reused ontologies should be transferred to the semantic data model using the ETL tool (see Section 5.3) described in the previous sections.

4. Test datasets should be created by selecting the concepts reused by reference ontology. It should be noted that the chosen concepts should not be the concepts defined by the reference ontology itself. Their definitions should

be in reused ontologies. Thus, we can try to create reference ontology using the concepts of reused ontologies.

5. Each dataset should be given to ontology module extraction method as input.
6. Each concept definition in extracted ontology module should be compared with its definition in reference ontology.

The evaluation steps are a little bit complicated, but especially, step 5 should be detailed to understand recall/precision better. In Figure 7.2, how this calculation is performed has been shown using a detailed representation, where O stands for reference ontology and R states for ontologies reused by O. The set of reference ontology O does not fully cover the set of ontologies R. Because reused ontologies may not be completely imported. So there is only intersection between them.

I, which is acronym for I*ntended Module*, represents the ontology module to be created by the user. However, in this evaluation, reference ontology R is used to determine the intended module since we cannot know what the user really wants. Therefore, intended module is the module that contains all definitions and relations of the concepts in ontology module which is extracted using the modularization method. These definitions and relations are taken from reference ontology R.

Suppose that, user is already building reference ontology R from scratch and creating modules manually or using a modularization tool and aims to create I. If they wanted to create this module using the method and the tool developed in scope of thesis, what would this module look like? This module is represented with M. If the compatibility of the module M with the module I can be measured using recall/precision calculations, success of reuse can be estimated. Reference ontology is used to specify the scope of intended module. Intended module contains all relations and definitions of concepts of module M as stated above.

It is important to understand how TP (true positives), FP (false positives) and FN (false negatives) are handled while evaluating. All definitions both in module M and module$I$ are true positives. The definitions only in module M are false positives, because they are not in module I. The definitions not found in module M, but already in module I are false negatives.

A sample is shown in Figure **??**to clarify this calculation. Suppose that,

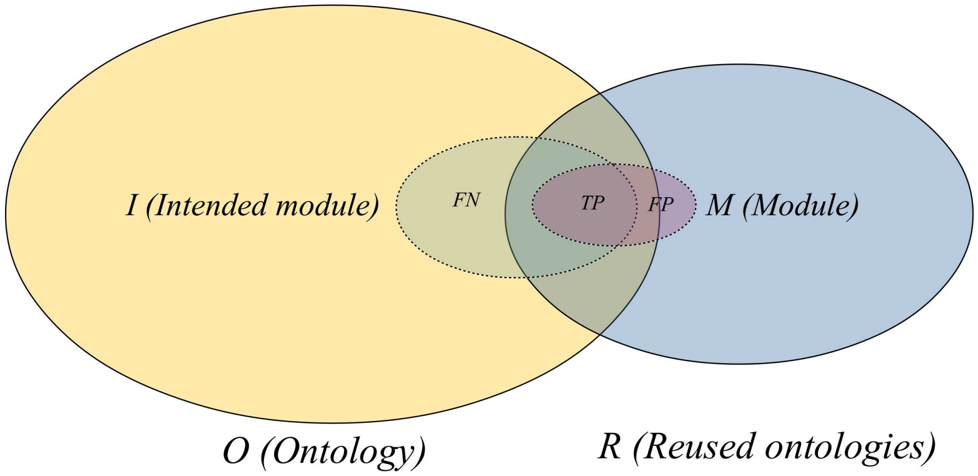


Figure 7.2: Recall/precision of ontology reuse.

user selected sign-recording and clinical-act classes from CPR ontology and extracted a module using the ontology modularization method implemented is thesis. RDF/XML representation of this module is shown in Listing 7.2.

```
<?xml version="1.0"?>
<rdf:RDF xmlns="#"
         xmlns:cpr="http://purl.org/cpr/"
         xmlns:ro="http://www.obofoundry.org/ro/ro.owl#"
         xmlns:bfo="http://www.ifomis.org/bfo/1.1/span#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

    <owl:ObjectProperty rdf:about="cpr:hasRole"/>
    <owl:ObjectProperty rdf:about="cpr:outputOf"/>
    <owl:ObjectProperty rdf:about="ro:has_agent"/>

    <owl:Class rdf:about="cpr:clinical-act">
        <rdfs:subClassOf rdf:resource="bfo:ProcessualEntity"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="ro:has_agent"/>
                <owl:someValuesFrom>
                    <owl:Restriction>
                        <owl:onProperty
                         ↪  rdf:resource="cpr:hasRole"/>
                        <owl:someValuesFrom rdf:resource=
                         ↪  "cpr:healthcare-professional-role"/>
                    </owl:Restriction>
                </owl:someValuesFrom>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty
                 ↪  rdf:resource="ro:has_participant"/>
                <owl:someValuesFrom rdf:resource="cpr:patient"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>

    <owl:Class rdf:about="cpr:clinical-finding">
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="cpr:outputOf"/>
                <owl:someValuesFrom
                 ↪  rdf:resource="cpr:clinical-act"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>

    <owl:Class rdf:about="cpr:healthcare-professional-role"/>

    <owl:Class rdf:about="cpr:patient">
    </owl:Class>

    <owl:Class rdf:about="cpr:sign-recording">
        <rdfs:subClassOf rdf:resource="cpr:clinical-finding"/>
    </owl:Class>

    <owl:Class rdf:about="bfo:ProcessualEntity">
    </owl:Class>

</rdf:RDF>
```

Listing 7.2: Extracted ontology module M for sign-recording and clinical-act

The module shown in Listing 7.2 corresponds to the module labeled with M. The scope of intended module is all definitions of concepts included in extracted module M. These definitions should be taken from reference ontology R. Intended module I for M is shown in Listing 7.3.

```
<rdf:RDF xmlns="#"
         xmlns:cpr="http://purl.org/cpr/"
         xmlns:ro="http://www.obofoundry.org/ro/ro.owl#"
         xmlns:bfo="http://www.ifomis.org/bfo/1.1/span#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:ObjectProperty rdf:about="cpr:hasRole">
        <rdfs:domain>
            <owl:Class rdf:about="bfo:IndependentContinuant"/>
        </rdfs:domain>
        <rdfs:range rdf:resource="bfo:Role"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="cpr:outputOf">
                <owl:inverseOf rdf:resource="cpr:hasOutput"/>
        <rdfs:range rdf:resource="bfo:ProcessualEntity"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="ro:has_agent">
    </owl:ObjectProperty>

    <owl:Class rdf:about="cpr:clinical-act">
        <rdfs:subClassOf>
            <owl:Class rdf:about="bfo:ProcessualEntity"/>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="ro:has_agent"/>
                <owl:someValuesFrom>
                    <owl:Restriction>
                        <owl:onProperty
                         ↪  rdf:resource="cpr:hasRole"/>
                        <owl:someValuesFrom rdf:resource=
                         ↪  "cpr:healthcare-professional-role"/>
                    </owl:Restriction>
                </owl:someValuesFrom>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty
                 ↪  rdf:resource="ro:has_participant"/>
                <owl:someValuesFrom rdf:resource="cpr:patient"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>

    <owl:Class rdf:about="cpr:clinical-finding">
        <owl:disjointWith rdf:resource="cpr:symptom-recording"/>
        <owl:disjointWith rdf:resource="cpr:patient-record"/>
        <owl:disjointWith
         ↪  rdf:resource="cpr:recorded-clinical-situation"/>
        <rdfs:subClassOf rdf:resource="cpr:clinical-artifact"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="cpr:outputOf"/>
                <owl:someValuesFrom
                 ↪  rdf:resource="cpr:clinical-act"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="cpr:composedBy"/>
                <owl:someValuesFrom>
                    <owl:Restriction>
                        <owl:onProperty
                         ↪  rdf:resource="cpr:hasRole"/>
                        <owl:someValuesFrom
                         ↪  rdf:resource="cpr:clinician-role"/>
                    </owl:Restriction>
                </owl:someValuesFrom>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty
                 ↪  rdf:resource="cpr:representationOf"/>
                <owl:someValuesFrom
                 ↪  rdf:resource="cpr:bodily-feature"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>

    <owl:Class rdf:about="cpr:healthcare-professional-role">
        <rdfs:subClassOf rdf:resource="bfo:Role"/>
    </owl:Class>
```

```
    <owl:Class rdf:about="cpr:patient">
        <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="cpr:person"/>
                    <owl:Restriction>
                        <owl:onProperty
                         ↪  rdf:resource="cpr:hasRole"/>
                        <owl:someValuesFrom
                         ↪  rdf:resource="cpr:patient-role"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty
                         ↪  rdf:resource="ro:participates_in"/>
                        <owl:someValuesFrom
                         ↪  rdf:resource="cpr:clinical-act"/>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:equivalentClass>
    </owl:Class>

    <owl:Class rdf:about="cpr:sign-recording">
        <owl:intersectionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="cpr:clinical-finding"/>
            <owl:Restriction>
                <owl:onProperty rdf:resource="cpr:outputOf"/>
                <owl:someValuesFrom
                 ↪  rdf:resource="cpr:clinical-examination"/>
            </owl:Restriction>
        </owl:intersectionOf>
        <rdfs:subClassOf rdf:resource="cpr:clinical-finding"/>
    </owl:Class>

        <owl:Class rdf:about="bfo:ProcessualEntity"/>

</rdf:RDF>
```

Listing 7.3: Intended ontology module I for sign-recording and clinical-act

In this sample, only relations explicitly shown in RDF/XML format (see Listing 7.2 and 7.3) used to calculate recall and precision to make it clear for readers. In actual calculations, whose results are given in Section 7.5, all implicit relations, such as *rdf:type*, *rdf:list*, *rdf:first*, and *rdf:rest*, that can be seen in N3 triple notation are used.

As seen in Listing 7.2 there are 18 class definitions, property definitions and relations. Corresponding intended module has 40 definitions for the same types. So, TP value is 18 and FN value is 22 for this module extraction. Recall can be calculated using formula $TP/(TP+FN)$ and recall value is $18/(18+22) = 0,35$. Because every definition of module M is also in intended module I, precision value is 1.

7.2 Sleep Domain Ontology: A Reference Ontology for Evaluation

Sleep Domain Ontology (SDO) is an ontology developed to semantically model the data obtained in sleep medicine (Arabandi et al., 2010). While SDO has already comprehensively addressed ontology reuse, it is very suitable to be used as a reference ontology to evaluate thesis study. It has used ontologies that contain large amounts of data along with smaller ontologies. Developers of this ontology has chosen partial reuse rather importing all ontology.

Another important feature of SDO is that it has been evaluated for reuse within the scope of important studies. In (Kamdar et al., 2017), (Ochs et al., 2017) and (Halper et al., 2018), SDO has been extensively discussed to evaluate current state of ontology reuse. The ontologies reused by SDO ontology are listed in Table 7.2.

Table 7.2: Ontologies reused by SDO.

Ontologies reused by SDO
Basic Formal Ontology (BFO)
Relations Ontology (RO)
Computer-Based Patient Record Ontology (CPR)
Information Artifact Ontology (IAO)
Ontology for General Medical Science (OGMS)
Units of Measurement Ontology (UO)
Biotop Ontology Foundational Model of Anatomy (FMA)

Hierarchy of ontologies reused by SDO is shown in Figure 7.3.

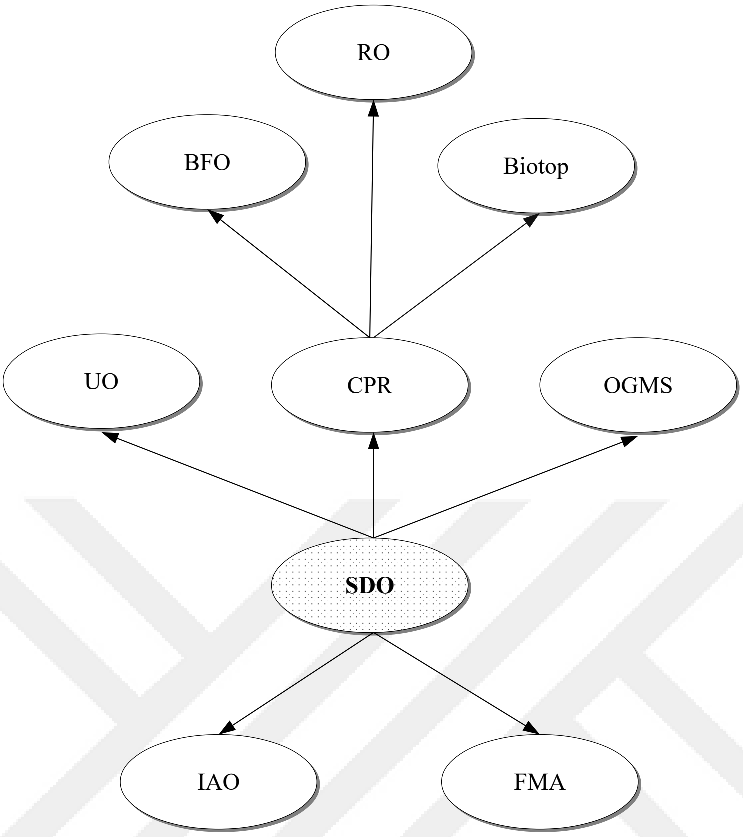


Figure 7.3: Hierarchy of ontologies reused by SDO.

SDO has been analyzed in detail. Number of reused classes and number of reused properties is given in Table 7.3. In evaluation studies, SDO has been used as a reference point for this thesis. All performance and recall/precision calculations are performed using SDO and its reuse catalog.

7.3 Building Test Environment

Once the SDO ontology has been analyzed and ontologies reused were identified, appropriate versions of these ontologies have been searched. However, the SDO does not explicitly specify which versions of ontologies are used. Considering the history of SDO ontology and the namespaces it uses, appropriate versions of the relevant ontologies were searched using repositories such as Bioportal. As a result, the nearest versions were obtained. However, the appropriate versions of some ontologies could not be found and current versions were preferred. In addition, FMA ontology is not fully compatible with the version used in SDO. This will cause the recall and precision values to drop slightly. Reused ontologies

Table 7.3: SDO reuse analysis.

Ontology	Number of reused classes	Number of reused properties
SDO	400	21
IAO	6	0
OGMS	71	0
CPR	78	27
Biotop	26	11
BFO	21	0
FMA	539	0
UO	10	11

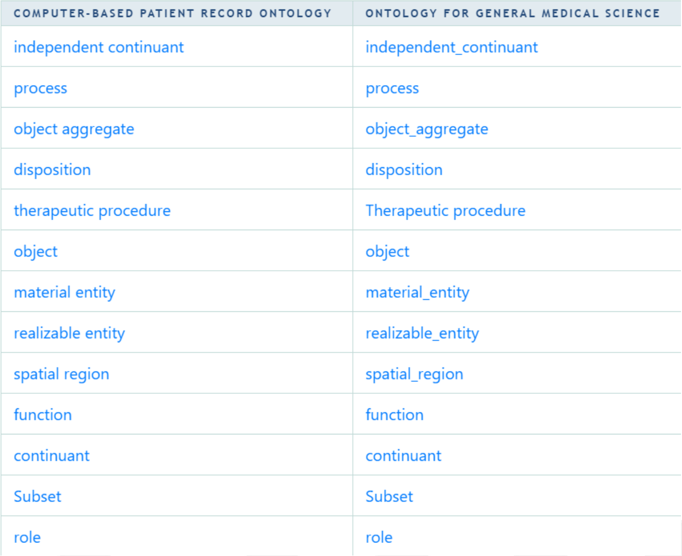
and their versions used in test environment are listed in Table 7.4.

Table 7.4: Versions of ontologies used in test environment.

Ontology	Version
BFO	07/2009
RO	11/2017
CPR	04/2013
IAO	09/2012
OGMS	02/2012
UO	2008
Biotop	11/2009
FMA	09/2009

These ontologies have been transferred to the semantic data model using the ETL tool detailed in Chapter 5.3. During the transfer, mappings between ontologies which are not explicitly defined in OWL files has been obtained using Bioportal mapping services. These services automatically detects equivalent classes between ontologies which are not explicitly defined (Salvadores et al., 2013). Some of the mappings (only equivalent classes) between CPR and OGMS ontology is shown Figure 7.4 as a sample. All mappings between used ontologies has also been transferred to test data set.

The test data sets created on this test environment are detailed in the following subsections.



COMPUTER-BASED PATIENT RECORD ONTOLOGY	ONTOLOGY FOR GENERAL MEDICAL SCIENCE
independent continuant	independent_continuant
process	process
object aggregate	object_aggregate
disposition	disposition
therapeutic procedure	Therapeutic procedure
object	object
material entity	material_entity
realizable entity	realizable_entity
spatial region	spatial_region
function	function
continuant	continuant
Subset	Subset
role	role

Figure 7.4: Mappings between CPR and OGMS (Bioportal).

7.3.1 DataSet-1

This data set has been created manually. In each sub-dataset, concepts which are thought to be more related to each other are selected. Thus, it is aimed to measure the reuse more accurately. While other data sets are only used for performance evaluation, this data set is also used for recall/precision evaluation.

An example from this data set is shown in Listing 7.4. As seen in the listing, the concepts are highly related to each other. For example, *clinical-diagnosis* and *clinician-role* selected together to form an ontology module.

```
<cpr:clinical-diagnosis>
<cpr:clinician-role>

<cpr:sign-recording>
<cpr:clinical-act>

<cpr:sign-recording>
<cpr:screening-act>
```

Listing 7.4: DataSet-1 sample.

All items of DataSet-1 is listed in Appendix 3.

7.3.2 DataSet-2

This data set has been created randomly and is only used for performance evaluation. The elements of each subset in this dataset has been selected based on the reuse weight of the ontology in SDO. If an ontology is reused more frequently in SDO, the elements of this ontology is included more frequently in DataSet-2. A sample part from this data set is shown in Listing 7.5. As seen in the listing, there are no direct relations between selected concepts.

```
<bfo:ProcessualEntity>
<cpr:anatomical-space>

<obo:IAO_0000078>
<fma:fma9576>
<cpr:sign-recording>

<cpr:laboratory-test-finding>
<cpr:anatomical-space>
<bfo:ScatteredSpatiotemporalRegion>
```

Listing 7.5: DataSet-2 sample.

All items of DataSet-2 is listed in Appendix 4.

7.4 Results of Performance Evaluation

Both datasets (DataSet-1 and DataSet-2) has been used in performance evaluations. The server configuration on which the tests are run is given in Table 7.5.

Table 7.5: Server configuration.

CPU	Intel Core i7 6700 HQ 2.60 Ghz
Architecture	64 bit
Memory	16GB
Disk	256GB SSD
OS	Windows 10

Each dataset has been run on 4 different application configurations. These configurations are based on the number of threads used and whether or not the cache is enabled. The details of the configurations are as follows:

- **Configuration 1:** Single thread, cache disabled
- **Configuration 2:** 10 threads, cache disabled
- **Configuration 3:** Single thread, cache enabled
- **Configuration 4:** 10 threads, cache enabled

All performance results (time, number of nodes, number of paths, etc.) indicate average values. For example; if there is more than one test set with 2 elements selected, the resulting value indicates the average for each result for 2 element sub-sets. The following sections compare all possible configurations for performance evaluation.

7.4.1 Single thread vs single thread and cache (DataSet-1)

If cache is used in path and Steiner Tree algorithms, it is seen that the performance increases significantly. When algorithms are run in single thread using DataSet-1, the performance increase due to cache usage is shown in the Figure 7.5. In most cases, using cache with single thread doubles the performance at least.

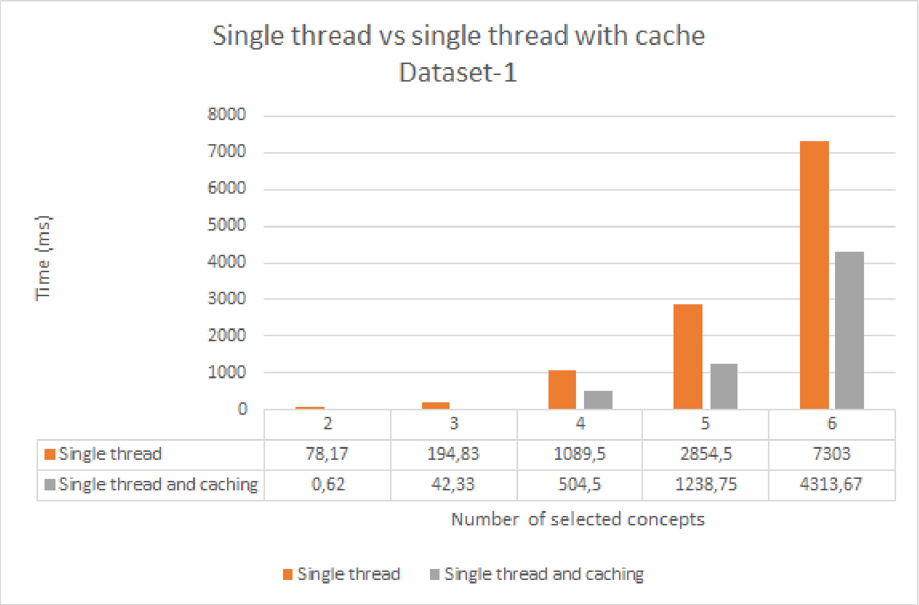


Figure 7.5: Performance comparison of single thread and caching in DataSet-1.

While DataSet-2 has been created randomly, it is expected that extraction of ontology modules will take longer when compared to DataSet-1, due to the low relatedness between concepts, and Figure 7.6 confirms this expectation. Since the execution of algorithm takes longer, the effect of cache usage has also increased in DataSet-2. In some cases, especially when the number of selected concepts is 5 or higher, there is 3 times performance increase while using cache in DataSet-2.

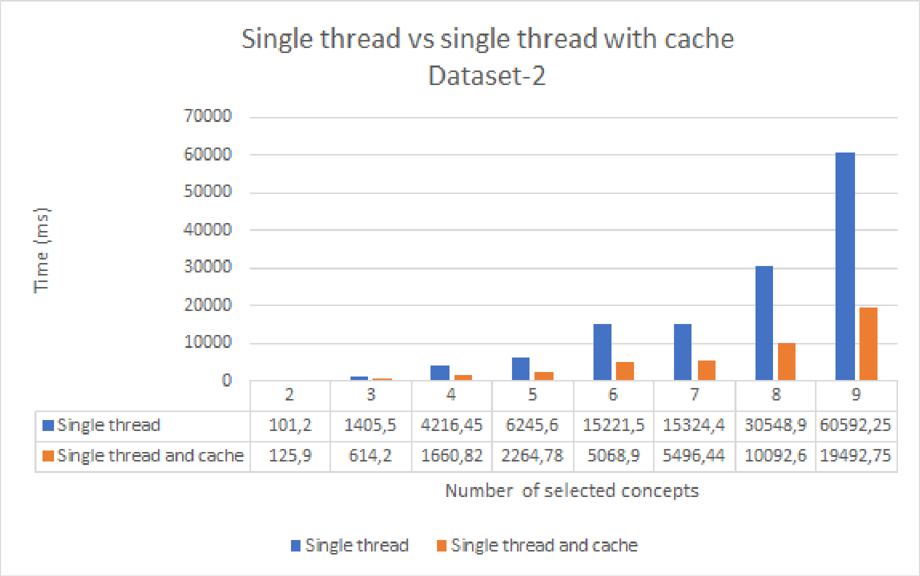


Figure 7.6: Performance comparison of single thread with cache in DataSet-1.

7.4.2 Single thread vs 10 threads

The increase in the number of threads also gives a significant performance enhancement as shown in Figure 7.7. As the number of selected concepts increases in DataSet-1, using more threads gives better results.

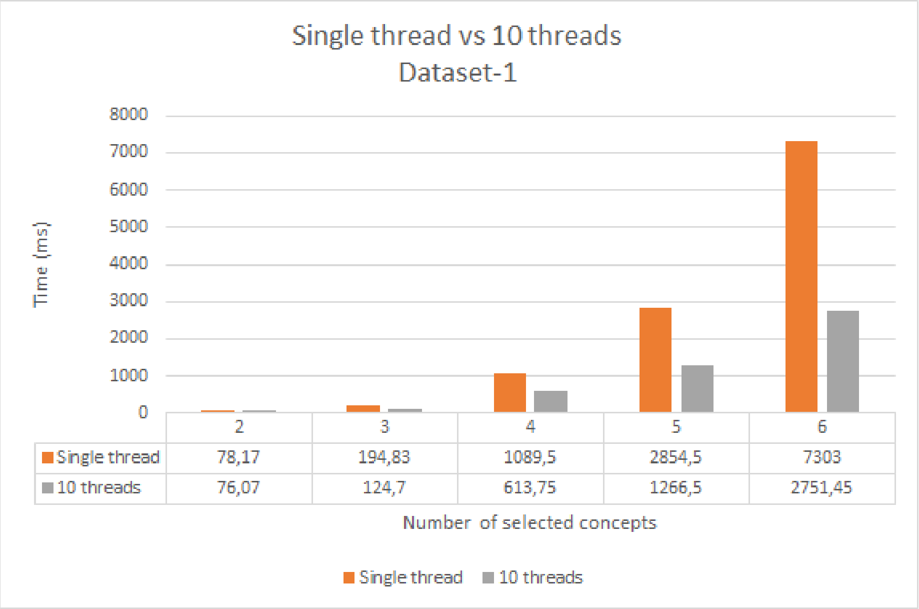


Figure 7.7: Performance comparison of single thread and 10 threads in DataSet-1.

Evaluation of DataSet-2 also gives similar results (see Figure 7.8). DataSet-2 has been created randomly and it is highly possible that relatedness of concepts is less than DataSet-1. Since the number of traversed paths and nodes increases, using more threads gives better results when compared to DataSet-1. However, when the number of selected concepts is 6 or higher, impact of thread usage decreases due to high number of visited paths and nodes.

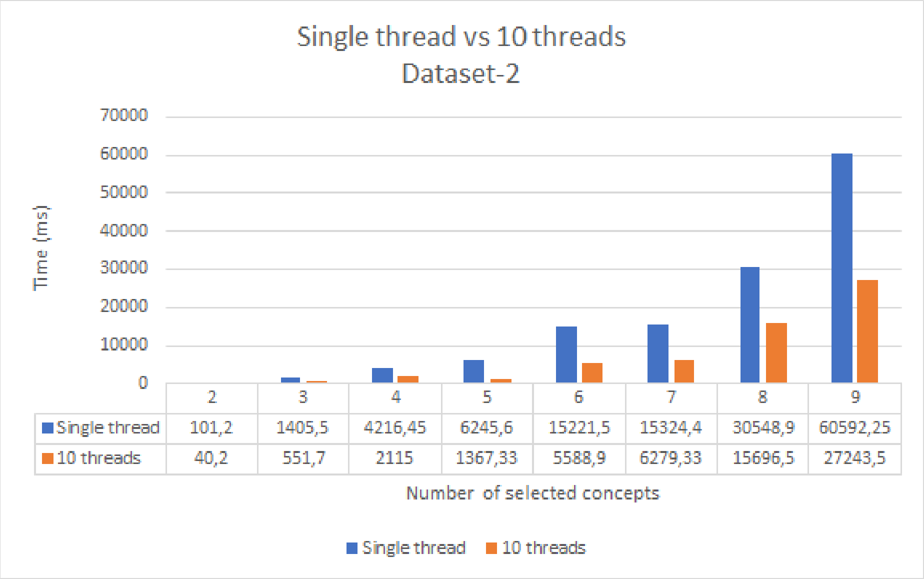


Figure 7.8: Performance comparison of single thread and 10 threads in DataSet-2.

7.4.3 10 threads vs 10 threads and caching

Only increasing the number of used threads without cache brings a significant performance enhancement as discussed in previous section. As seen in Figure 7.9 and Figure 7.10, cache usage also provides performance increase in cases where high number of threads are used.

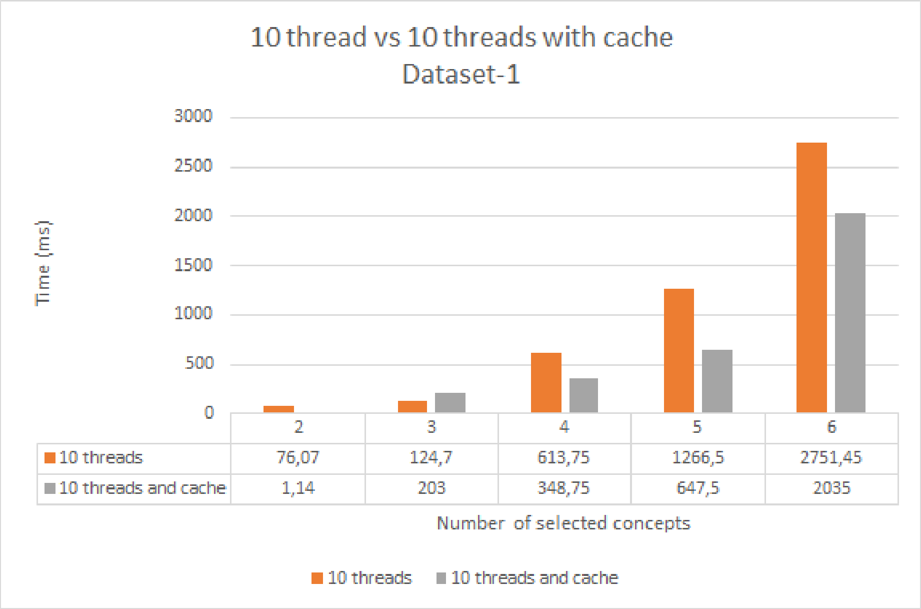
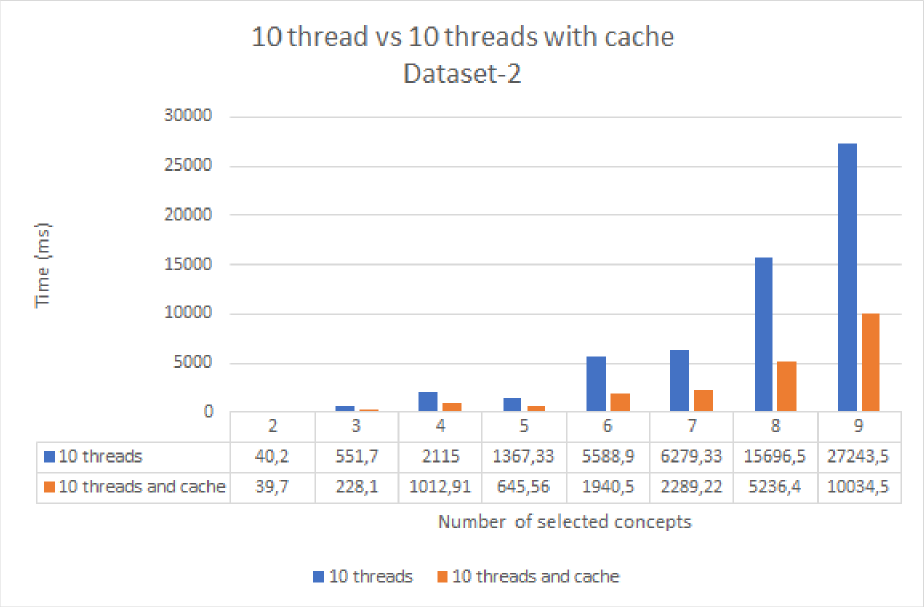


Figure 7.9: Performance comparison of 10 threads and 10 threads with cache in DataSet-1.

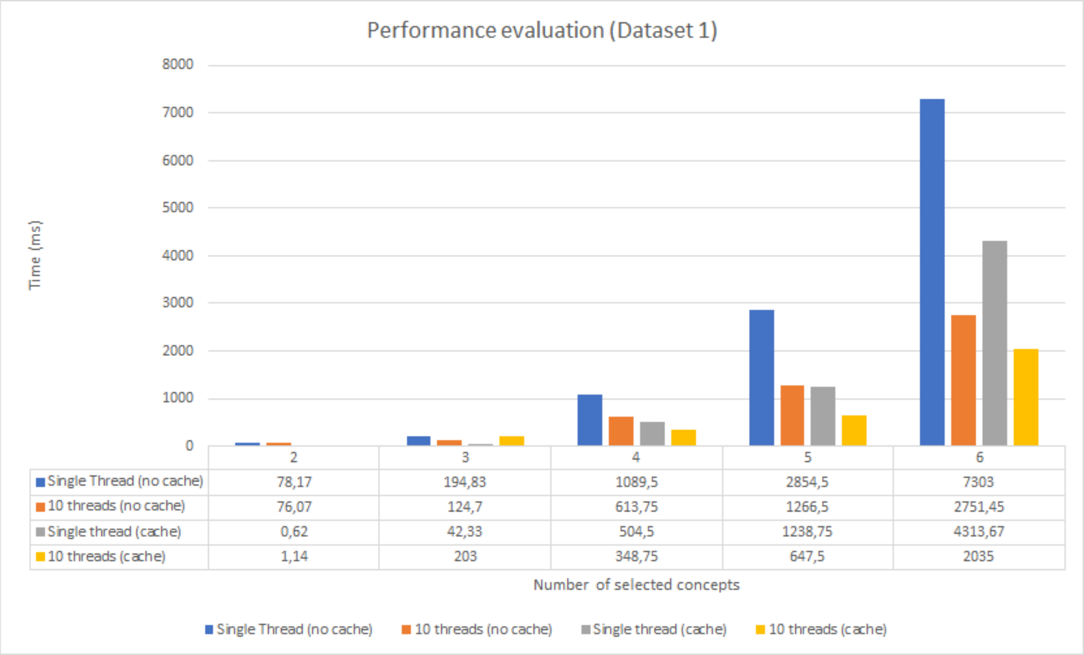
As in previous comparisons, while relatedness between concepts in DataSet-2 is low, effectiveness of using cache is higher when compared to DataSet-1.



	2	3	4	5	6	7	8	9
🟦 10 threads	40,2	551,7	2115	1367,33	5588,9	6279,33	15696,5	27243,5
🟧 10 threads and cache	39,7	228,1	1012,91	645,56	1940,5	2289,22	5236,4	10034,5

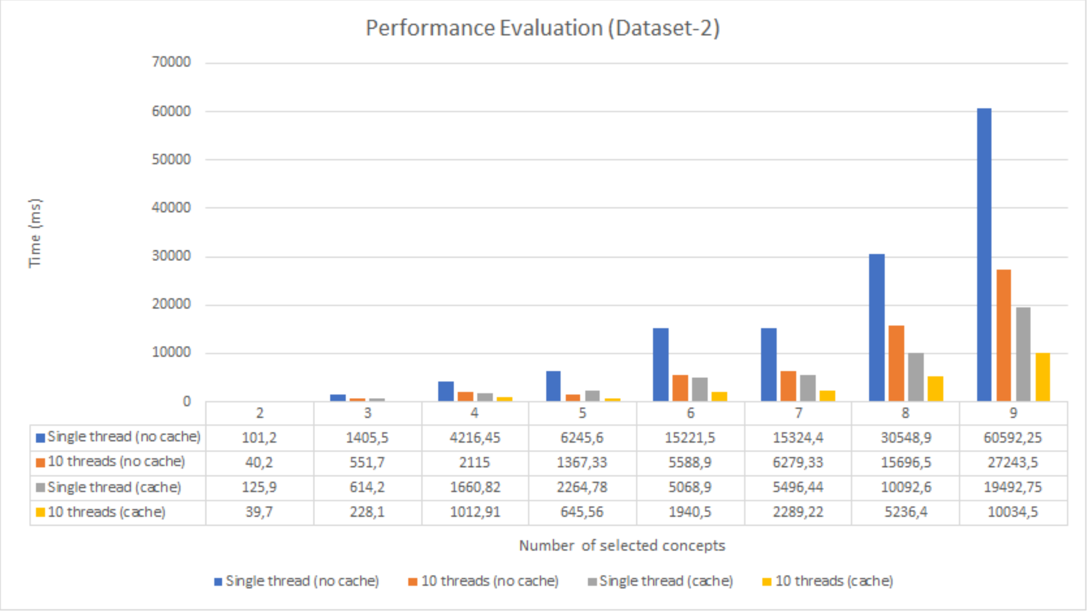
Figure 7.10: Performance comparison of 10 threads and 10 threads with cache in DataSet-2.

Although different alternatives are compared separately, an overall comparison should also be given to see the best result. As seen in Figure 7.11 and Figure 7.12, using cache with high number of threads gives best results nearly for all conditions. The cost of algorithm, when using a single thread without cache, becomes more evident as the number of selected concepts increases.



	2	3	4	5	6
🟦 Single Thread (no cache)	78,17	194,83	1089,5	2854,5	7303
🟧 10 threads (no cache)	76,07	124,7	613,75	1266,5	2751,45
⬜ Single thread (cache)	0,62	42,33	504,5	1238,75	4313,67
🟨 10 threads (cache)	1,14	203	348,75	647,5	2035

Figure 7.11: Performance comparison of all alternatives in DataSet-1.

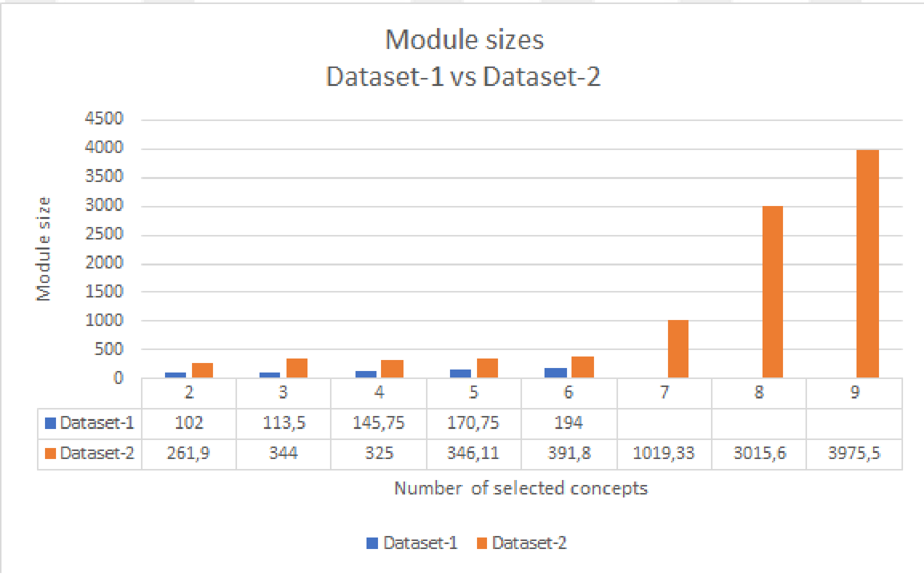


	2	3	4	5	6	7	8	9
🟦 Single thread (no cache)	101,2	1405,5	4216,45	6245,6	15221,5	15324,4	30548,9	60592,25
🟧 10 threads (no cache)	40,2	551,7	2115	1367,33	5588,9	6279,33	15696,5	27243,5
⬜ Single thread (cache)	125,9	614,2	1660,82	2264,78	5068,9	5496,44	10092,6	19492,75
🟨 10 threads (cache)	39,7	228,1	1012,91	645,56	1940,5	2289,22	5236,4	10034,5

Figure 7.12: Performance comparison of all alternatives in DataSet-2.

7.4.4 Comparison of module sizes in DataSet-1 and DataSet-2

When the number of modules is examined, it is seen that there is not a significant increase up to 6 concepts for both data sets (see Figure 7.13). As the relatedness between the concepts in the DataSet-2 is less, the module size is higher as expected. Since there are no more than 6 concepts in the DataSet-1, it is not possible to see results for more concepts in this data set. However, when more than 6 concepts are selected in the DataSet-2, the increase in module size becomes more evident.



	2	3	4	5	6	7	8	9
Dataset-1	102	113,5	145,75	170,75	194			
Dataset-2	261,9	344	325	346,11	391,8	1019,33	3015,6	3975,5

Figure 7.13: Comparison of module sizes in DataSet-1 and DataSet-2.

7.4.5 Comparison of number of visited paths in DataSet-1 and DataSet-2

As in the comparison of the module sizes, the number of visited paths is higher for the DataSet-2. However, the increase in the number of paths is more consistent when compared to module sizes, and there is no significant difference for specific number of selected concepts as see in Figure 7.14.

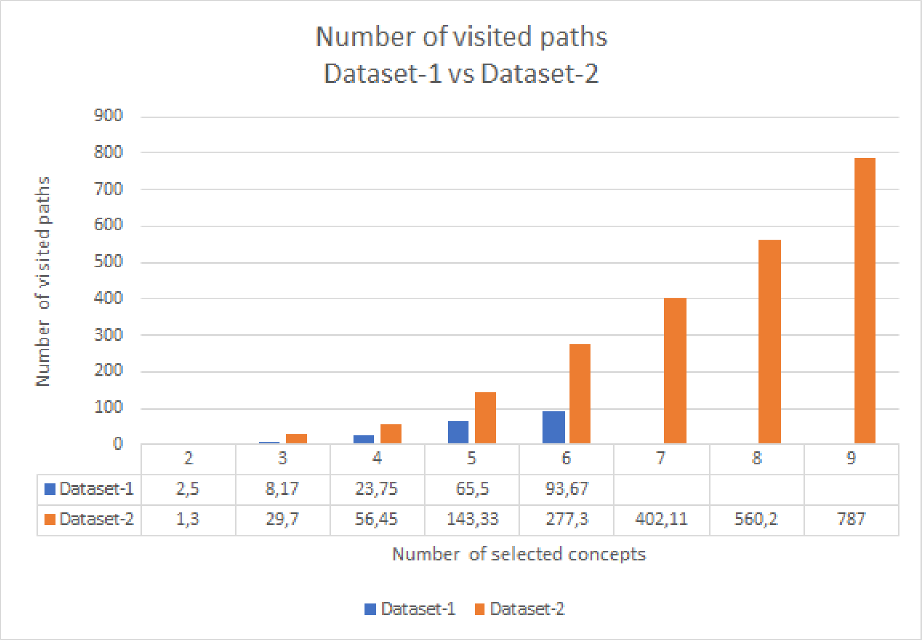


Figure 7.14: Comparison of number of visited paths in DataSet-1 and DataSet-2.

7.4.6 Comparison of number of visited nodes in DataSet-1 and DataSet-2

As seen in Figure 7.15, the number of visited nodes is higher for the DataSet-2 as in the number of visited paths.

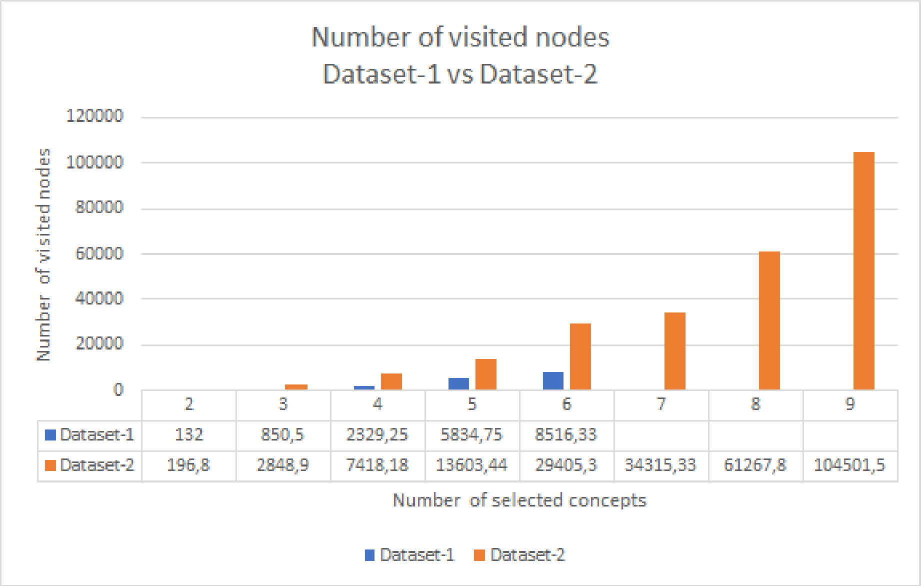


Figure 7.15: Comparison of number of visited nodes in DataSet-1 and DataSet-2.

7.4.7 Conclusion

Performance comparisons revealed some interesting results. It is possible to summarize these comparisons, which provide information about the basic behaviors of the ontology modularization method, as follows:

- Increase in response time is more evident when the number of selected concepts is more than 6.
- Increasing the number of threads and using cache provides a significant performance enhancement.
- The best performance is achieved by using multiple threads and cache together.
- Since the relatedness between concepts is less in randomly selected data set (DataSet-2), performance is negatively affected. However, the effect of using thread and cache increases performance better in DataSet-2.
- As the relatedness between concepts decreases, module size increases.
- If the number of concepts selected is more than 6, the increase in module size becomes more evident. However, this may change for different data sets.

- As the relatedness between concepts decreases, number of visited nodes and paths increase.

7.5 Results of Recall/Precision Evaluation

The recall/precision calculations have only been run on DataSet-1, since this data set has been constructed manually from an ontology engineer perspective. Thus, it is thought that more realistic results can be obtained. How recall and precision are calculated detailed in Section 7.2.

As mentioned in the previous sections, SDO has been used for comparisons. Recall/precision calculations has been performed according to two different configurations. The following parameters are used to create these configurations:

- Include all super-classes recursively
- Include all equivalent classes
- Include all properties of classes
- Include all domain/range definitions of properties

In the first configuration, all parameter values are given as false. Thus, syntactically and semantically valid minimal modules are created. Therefore, especially the recall value is expected to be lower. The values obtained with these parameters are given in Table 7.6.

In the second configuration, all parameters are set to true. The modules are created in the largest possible form. Since the module contains more details, the recall values obtained from its comparison with the reference ontology will be higher. The results observed by using these parameters are given in Table 7.7.

As shown in the Table 7.6 and Table 7.7, there is more than one result for each “number of selected concepts” value. For example; for the case where the number of selected concepts is 2, 5 different results are given. This is due to the examples in data sets. There are 5 examples in the DataSet-1 where the number of concepts is 2 (see Appendix 3 for details).

Table 7.6: Recall/precision values (all parameters false).

#Concepts	Recall	Precision	#Concepts	Recall	Precision
2	0.2475	0.8929	4	0.875	0.8818
2	0.1607	0.75	4	0.9773	0.9113
2	0.2857	0.4	4	0.9302	0.9071
2	0.2295	0.9333	4	0.9459	0.9130
2	0.2569	0.9655	5	0.8254	0.5204
3	0.2456	0.875	5	0.9091	0.5175
3	0.2826	0.7647	5	0.8387	0.8729
3	0.2793	0.9118	5	1.0	0.9147
3	0.2755	0.9310	6	0.9643	0.9022
3	0.2558	0.7857	6	0.8868	0.5157
3	0.3191	0.9667	6	1.0	0.9214
4	0.3537	0.9677			
Recall	**Min**	0,1607	**Precision**	**Min**	0,4
	Max	0,458		**Max**	1.0
	Median	0,3361		**Median**	0,9118
	Average	0,3222		**Average**	0,8824

7.6 Results

In this section, the proposed method and tool are evaluated according to the performance and reuse criteria. Although performance measurements has been performed on both data sets, recall/precision evaluations has only been run on manually generated DataSet-1 to observe more realistic results.

Performance evaluations has been performed according to parameters of response time, number of traversed graph nodes, number of traversed paths and module size. The main purpose of these evaluations is to demonstrate the usability of the method as a semi-automatic approach. It is also important to prove that scalability can be achieved through configuration update. Although it is not intended to compare different methods or different data sets, some interesting results have also been observed. In randomly created data sets, performance of method is worse. This is due to the low relatedness between selected concepts. With low relatedness, it is highly possible that concepts are far away from each other and the number of traversed paths and nodes are higher to connect these concepts. It is also important to note that these evaluations are dependent to many factors, such as size of ontology, location of selected concepts and branching factor. Therefore, for

Table 7.7: Recall/precision values (all parameters true)

#Concepts	Recall	Precision	#Concepts	Recall	Precision
2	0.5786	0.9	4	0.7638	0.8818
2	0.5205	0.4436	4	0.7958	0.9113
2	0.6786	0.8769	4	0.7651	0.9071
2	0.5989	0.436	4	0.7875	0.9130
2	0.6448	0.4538	5	0.7615	0.5204
3	0.6312	0.9175	5	0.7617	0.5175
3	0.6744	0.8657	5	0.7744	0.8729
3	0.6404	0.4488	5	0.8027	0.9147
3	0.6312	0.9175	6	0.8054	0.9022
3	0.7064	0.8953	6	0.7628	0.5157
3	0.6875	0.9263	6	0.7914	0.9214
4	0.775	0.8942			
Recall	**Min**	0.5205	**Precision**	**Min**	0.436
	Max	0.8054		**Max**	0.9263
	Median	0.7615		**Median**	0.8942
	Average	0.7104		**Average**	0.7719

different data sets and ontologies, different results can be observed.

Evaluations has been made according to 4 different configurations for these purposes. When we examine the performance values, it is observed that response times are within acceptable limits for semi-automated tool. Especially, if 6 or more concepts are selected, the increase in response time is getting more significant. However, increasing the number of threads and enabling the use of cache dramatically reduces the response time. These performance enhancements has been observed even though only Java threads are used during these tests. However, it is thought that better results can be achieved by creating a distributed architecture, using multiple machines and using a distributed application manager such as Apache Akka[46]. Because the structure of the method is very suitable for this kind of distributed architecture.

The other results obtained for performance are number of traversed nodes, number of traversed paths and module sizes. Although these results have no direct effect on use, they included in order to make a better evaluation of proposed method.

[46] https://akka.io/

Recall/precision calculations are performed to show how successful the extracted modules are according to a reference ontology. The recall value increases when all parameters are set to true because the extracted modules are getting bigger and contain more information. While the average recall value is 0.7104 when all parameters are true, the average recall value is 0.3222 when the parameters are set to false. These values are considered to be sufficient for a semi-automatic method. On the other hand, precision values are not always 1. Because the same versions of the ontologies used by SDO cannot be found, and extracted modules can sometimes contain information which are not included in SDO.

8. CONCLUSION AND FUTURE WORKS

In this thesis, studies on the development of a method and a tool to facilitate the reuse of ontologies have been carried out. The proposed method has been implemented on semantic cloud architecture and client applications, which make use of reuse method easier, have been designed and developed. In addition, a reuse methodology has been proposed to support the reuse process.

Ontology reuse is one of the most challenging issues in the ontology development literature and is still seen as an unsolved problem. Many sub-fields such as ontology evaluation, ontology modularization, ontology search, ontology mapping, ontology alignment and ontology development methodologies are topics that need to be addressed in the context of ontology reuse. As the scope is so broad, there is no study that considers the reuse problem as a whole. As a result, ontologies can not be reused at the desired level, and development costs increase. Most importantly, modeling the same information multiple times by different ontologies brings inconsistency. Kamdar et al. (2016) and Ochs et al. (2017) examined the current state of ontology reuse in their work. The most important point highlighted by Kamdar et al. (2016) is the need for a semi-automatic method and tool supporting ontology reuse, and this point also constitutes the aim of the thesis since the very beginning.

Throughout the thesis study, many sub-fields of ontology reuse have been studied in detail. It is possible to summarize these topics as follows:

- What ontology reuse is and current problems
- Ontology development methodologies and how they handle ontology reuse
- Ontology modularization methods
- Ontology reuse tools and applications
- How to access existing ontologies (ontology repositories and other sources)

With the clarification of the requirements, studies have been started on the method and tool that support ontology reuse. In these studies, the topics summarized below are especially emphasized:

- An ontology development methodology
  - An iterative reuse approach which is integrated with ontology development life cycle
- Path finding
  - BFS, DFS, Bidirectional search
  - Semantic relatedness
  - Semantically correct paths
- Ontology modularization
  - Steiner tree problem
  - Performance evaluation of Steiner tree methods
  - Syntactic completeness
  - Semantic completeness
- Semantic cloud architecture
  - NoSQL semantic data model (Neo4j, MongoDb, ElasticSearch)
  - Semantic cloud services (REST endpoints)
- Ontology reuse tools
  - Web application
  - Protege plugin

The evaluation of the studies and their comparison with the existing methods has already been carried out in the conclusion subsections of each section. In this section, where the whole thesis is evaluated, the previous evaluations will be summarized briefly.

In order to determine the requirements of the reuse process and modularization method to be developed better, a reuse-based methodology has been defined. This methodology is called Agile Ontology Development 101, inspired by the Ontology Development 101 (Noy and McGuinness, 2001) study. Agile Ontology Development 101 consists of multiple sub-iterations and includes ontology reuse as 2 sub-iterations in detail. Unlike other methodologies, in these two iterations, it details how reuse should be handled at different levels. It does not provide any guidance on project management and leaves the details to ontology engineers. A comparison has been made with other methodologies, especially in terms of reuse capabilities. This comparison is presented in Table 8.1.

Ontology modularization is an important step also to reuse large-scale ontologies. In this study, an ontology modularization method that can be integrated iteratively with the reuse process has been developed. This method is not intended to extract comprehensive ontology modules like many existing methods in the literature. Instead, it integrates the concepts chosen by the user using the semantic relatedness and creates a minimal module. Thus, users can extract modules repeatedly and integrate these modules in an iterative way and continue to the ontology development. Many methods of ontology and graph literature have been used to develop the ontology modularization method.. First of all, bidirectional BFS algorithm, which is integrated with semantic relatedness, has been used in path traversals. In addition, semantically correct path filters have been implemented. Besides, Steiner Tree problem has been used for modularization. Thus, the modules, in which the selected concepts are integrated in the most semantic way, are tried to be extracted. This feature puts the method ahead of the existing methods and makes it more suitable for ontology reuse. Comparison with existing methods in the literature is given in Table 8.2.

The implementation of the proposed method has been done on the semantic cloud architecture. A polyglot data architecture consisting of graph, document and full-text models has been created. A service-based development has been implemented on this data architecture. The development of the method on such an application architecture provides an important advantage over other available methods. In addition, all service methods are opened with REST endpoints. Two clients, a web application and a Protege plugin, have been developed using these REST endpoints. While these client applications, especially Protege plugin, are integrated with modularization method and reuse process, most of the problems of reuse tools have also been solved. Comparison of our reuse tool with existing tools is summarized in Table 8.3.

The absence of an ontology reuse evaluation method in the literature has been challenging in the evaluation of this study. For this reason, it is preferred to evaluate the ontology modularization method. However, the same problem has

Table 8.1: Comparison of Ontology Development 101 ontology development methodologies.

Methodology	**Life-cycle model**	**Project management**	**Is agile?**	**Ontology reuse**	**Details of ontology reuse**	**Location of ontology reuse**	**Tool/plugin support**
Grüninge & Fox	Not specified	No	No	Not specified	Not specified	Not specified	Not specified
Uschold & King	Not specified	No	No	Specified	Undetailed	Seperate step	On-to-knowledge
Methontology	Evolving prototypes	Yes	No	Specified	Undetailed	Supporting step	ODE
Ontology Development 101	Iterative	No	No	Specified	Undetailed	Seperate step	Not specified
Diligent	Iterative	No	No	Not specified	Not specified	Not specified	Not specified
Neon	Optional (iterative or waterfall)	No	No	Specified	Detailed	All methodology is based on reuse	Neon Toolkit
UPON Lite	Iterative	No	No	Not specified	Not specified	Not specified	Not specified
SAMOD	Iterative	No	Yes	Specified	Undetailed	Seperate step	Not specified
Agile Ontology Development 101 (Thesis study)	***Iterative***	***No***	***Yes***	***Specified***	***Detailed***	***Sub-iteration***	***Web application and protege plugin***

Table 8.2: Comparison of modularization method with existing methods.

	Type	**Concept selection/filtering**	**Property selection/filtering**	**Recursive class hierarchy**	**Semantic completeness**	**Semantic Relatedness**	**Reuse focused**	**GUI support**	**API support**	**Iterative**	**Methodology integration**
Noy & Musen	Traversal-based	Multiple	Yes	Yes	Yes	No	No	Yes	No	No	No
Seidenberg & Rector	Traversal-based	Multiple	Yes	Yes	Yes	No	No	No	No	No	No
Doran et al.	Traversal-based	Single	No	Yes	Yes	No	No	Yes	No	No	Yes
MOVE	Traversal-based	Multiple	Yes	Yes	Yes	No	No	No	No	No	No
Miao et al.	Graph-based	Multiple	No	No	No	No	No	No	No	No	No
SOMET	Query-based	Multiple	Yes	N/A	N/A	No	No	No	No	No	N/A
Hussain & Abidi	Traversal-based	Multiple	No	No	Yes	No	No	No	No	No	No
Ranwez at al.	Graph-based	Multiple	No	No	No	No	No	No	No	No	No
Abis (2011)	Graph-based	Multiple	No	No	Yes	Yes	No	Yes	No	No	No
Thesis Study	***Graph-based***	***Multiple***	***No***	***Optional***	***Limited & Optional***	***Yes***	***Yes***	***Yes***	***Yes***	***Yes***	***Yes***

Table 8.3: Comparison of reuse tool with existing tools.

	Status	**Type**	**Web API**	**Built-in Database**	**Graph-based GUI Support**	**Ontology Module Support**	**Methodology support**	**Extracting class definition**
MIREOT	Active	Plugin	No	No	No	No	No	MIREOT Principles
OntoFox	Active	Standalone	Yes	Yes	No	Yes	No	MIREOT Principles
Bioportal Import Plugin	Not Active	Plugin	Yes	Yes	No	No	No	MIREOT Principles
Protege Lov	Not Active	Plugin	Yes	Yes	No	No	No	No
Thesis Study	***Active***	***Plugin and Web Application***	***Yes***	***Yes***	***Yes***	***Yes***	***Yes***	***Semantic relatedness and parameter-based completeness***

been encountered here because the module requirements are completely subjective and there is no answer to the question of what a good module is. As a result, the recall/precision based measurement method proposed by Doran et al. (2007) has been adapted to our study. To obtain more accurate results, a reference ontology has been used with reverse engineering activities.

As a result of the measurements, it can be seen that the performance of the method is sufficient. Although there is a noticeable decrease in performance as the number of concepts increases, performance can be increased again with cache usage and other improvements.

As a result of this study, it has been focused on the development of a method and a tool that deal with the problems in the ontology reuse process as a whole and try to find a solution. The absence of a similar study in the literature is important in terms of the output obtained, although it creates difficulties in the evaluation phase.

Lastly, it is very important to emphasize which problems of reuse has been solved with this thesis study. In Section 2.3.3, we have already listed reuse problems and their possible solutions. How these problems are solved in thesis study is detailed in 8.4.

8.1 Future Works

In this study, although the ontology reuse process is considered as a whole, there are many sub-steps that are not included in the process. Both these sub-steps and what needs to be done to improve the current process are detailed in this section. In this context, future studies are detailed below:

- **Ontology evaluation:** Currently, the concepts, that users are looking for, are simply listed using search services. Suggestions such as which ontologies are more reliable, and if the same concept is included in more than one ontology, which one may be more appropriate to choose, are not presented. Such features are the subject of the ontology selection/recommendation and are important for improving the quality of the reuse process. (Jonquet et al., 2010), (Martínez-Romero et al., 2014) and (Martínez-Romero et al., 2017) are important studies in this subject. It is

Table 8.4: How thesis study solves reuse problems.

Problem	Status	Thesis solution
Low reuse rates	Solved	The defined methodology, process, method and tool will provide a solution to reuse problems. Therefore, making it available to end-users may increase reuse rates.
Mismatch and misunderstanding	Solved	The methodology and reuse process clearly defines all steps of reuse.
Finding right ontology	Solved	The search capabilities provided by the ontology reuse tool make it easier to find ontologies.
Ontology does not fit	Unsolved	This problem is not in the scope of thesis
Modularity	Solved	Implemented modularization method solves the modularity problem
Integration	Unsolved	Integration problem is not solved. But, ontology reuse tool can be extended to provide a feature to integrate reused concepts to developed ontology.
Development from scratch	Solved	The defined methodology, process, method and tool solve most of the reuse problem. Therefore, reusing existing ontologies do not bring extra reuse costs.

planned to integrate a method that can evaluate the ontologies searched in future studies. Thus, improvement in ontology reuse processes can be increased.

- **Portal:** Although the method has been developed on the semantic cloud architecture and various client tools have been presented, it is an important deficiency that there is no web portal for the end user. Users can view integrated ontologies and access detailed information through a portal.

- **Personalization:** The system should be able to improve its processes depending on the user's preferences and give more personalized results as a result . It is planned to carry out such a feature in the future.

- **Evaluation of Agile Ontology Development 101:** As mentioned earlier, an evaluation work has not been performed for Agile Ontology Development 101. In future studies, it is planned to use this methodology within the scope of an project and to make the necessary improvements by collecting the feedbacks from the participants.

- **Improving Protege plugin features:** The developed Protege plugin should be improved and expanded with the addition of new features.

- **Increasing the number of exported REST service methods:** The exported REST service methods are limited to the needs of the Protege plugin and web application. However, if these endpoints are used by external applications, they may not fully meet the requirements. The number of services exported as REST should be increased.

- **Adding a SPARQL endpoint and integration with search operations:** Currently, search functions are only available through full-text search capabilities. However, there may be situations where users need to search by queries. For such needs, a SPARQL endpoint should be presented and integrated with the search engine. However, Neo4j database used in the current data model does not support SPARQL. Therefore, ontology data must be stored simultaneously on an ontology database which supports SPARQL endpoints.

- **Automation of the ETL tool and integrating new ontologies automatically:** The transfer of ontologies to the semantic data model is provided by an ETL tool developed in scope of thesis study, as mentioned in the previous sections. This tool is used manually in its current form and sometimes it is required to do some modifications for specific ontologies when necessary. The ETL tool should also be automated so that new ontologies can be automatically transferred to the system.

- **Graph database scalability:** Neo4j graph database used in the semantic data model has the ability to scale horizontally. The Neo4j database is used with the embedded mode within the scope of the thesis study, and each working ontology modularization service contains an instance of the graph database in its own process. Therefore, scaling up the services horizontally causes multiplication of a new graph database instance, despite it is not required. To prevent this, ontologies stored on Neo4j should be distributed by type. Thus, the contents of different ontologies can work integrated within different services. However, some improvements are required in traversal algorithms for this improvement.

- **Improving distributed architecture:** Distribution is provided with threads in its current form. In this context, improvements are planned with a tool such as Apache Akka.

REFERENCES

Abiş, C.: 2011, Examination of sub-ontology extraction methods and development of a semantic relatedness based sub-ontology extraction method (in turkish: Alt ontoloji çıkarma yöntemlerinin incelenmesi ve anlamsal ilintililik tabanlı bir alt ontoloji çıkarma yönteminin geliştirilmesi), *Master's thesis*, Ege University, TR

Abiş, C., Giray, G., and Ünalır, M. O.: 2018, Implementing a cloud-based software architecture for ontology reuse (in turkish: Ontoloji yeniden kullanımı için bulut tabanlı bir yazılım mimarisinin gerçekleştirilmesi), in *12. Ulusal Yazılım Mühendisliği Sempozyumu*

Abiş, C., Giray, G., and Ünalır, M. O.: 2019, Agile ontology development 101: A reuse-based agile ontology development methodology (in turkish: Çevik ontoloji geliştirme 101: Yeniden kullanım tabanlı Çevik bir ontoloji geliştirme yöntemi), in *13. Ulusal Yazılım Mühendisliği Sempozyumu*, 1–12 pp

Allegrograph: 2020, *Allegrograph*, https://franz.com/agraph/allegrograph/, Last accessed on 26.01.2020

Arabandi, S., Ogbuji, C., and Redline, S.: 2010, Developing a sleep domain ontology, in *AMIA Clin. Res. Inform. Summit*

Arpírez, J. C., Gómez-Pérez, A., Lozano-Tello, A., and Pinto, H. S. A. N. P.: 2000, Reference Ontology and (ONTO)2 Agent: The Ontology Yellow Pages, *Knowledge and Information Systems* **2(4)**, 387

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z.: 2007, Dbpedia: A nucleus for a web of open data, in K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux (eds.), *The Semantic Web*, Berlin, Heidelberg, Springer Berlin Heidelberg, 722–735 pp

Balakirsky, S.: 2015, Ontology based action planning and verification for agile manufacturing, *Robotics and Computer-Integrated Manufacturing* **33**, 21 , Special Issue on Knowledge Driven Robotics and Manufacturing

Banker, K., Bakkum, P., Verch, S., Garrett, D., and Hawkins, T.: 2016, *MongoDB in Action*, Manning

Barthelemy, M., Chow, E., and Eliassi-Rad, T.: 2005, Knowledge representation issues in semantic graphs for relationship detection., in *AAAI Spring Symposium: AI Technologies for Homeland Security*, AAAI, 91-98 pp

Berners-Lee, T.: 2015, *5-star Open Data*

Berners-Lee, T., Hendler, J., and Lassila, O.: 2001, The Semantic Web, *Scientific American* **284(5)**, 28

Bhatt, M., Flahive, A., Wouters, C., Rahayu, W., and Taniar, D.: 2006, Move: A distributed framework for materialized ontology view extraction, *Algorithmica* **45(3)**, 457

Blazegraph: 2017, *Blazegraph Database*, https://wiki.blazegraph.com/wiki/index.php/Main_Page, Last accessed on 26.01.2020

BlazegraphGASAPI: 2016, *RDF GAS API*, https://wiki.blazegraph.com/wiki/index.php/RDF_GAS_API, Last accessed on 26.01.2020

Bontas, E. P., Mochol, M., and Tolksdorf, R.: 2005, Case studies on ontology reuse, in *In Proc. of the 5th International Conference on Knowledge Management*

Borst, W.: 1997, *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*, *Ph.D. thesis*, University of Twente, Netherlands

Brickley, D. and Guha, R.: 2014, *RDF Schema 1.1 - W3C Recommendation 25 February 2014*

Coffman, T., Greenblatt, S., and Marcus, S.: 2004, Graph-based Technologies for Intelligence Analysis, *Commun. ACM* **47(3)**, 45

Courtot, M., Gibson, F., Lister, A., Malone, J., Schober, D., Brinkman, R., and Ruttenberg, A.: 2011, Mireot: the minimum information to reference an external ontology term, *Applied Ontology* **6**, 23

Cyganiak, R., Wood, D., and Lanthaler, M.: 2014, *RDF 1.1 Concepts and Abstract Syntax*

d'Aquin, M. and Noy, N. F.: 2012, Where to publish and find ontologies? a survey of ontology libraries, *Web Semant.* **11**, 96

d'Aquin, M., Schlicht, A., Stuckenschmidt, H., and Sabou, M.: 2009, *Criteria and Evaluation for Ontology Modularization Techniques*, 67–89 pp, Springer Berlin Heidelberg, Berlin, Heidelberg

De Giacomo, G. and Lenzerini, M.: 1996, Tbox and abox reasoning in expressive description logics, Vol. 1996, 37-48 pp

De Nicola, A. and Missikoff, M.: 2016, A lightweight methodology for rapid ontology engineering, *Commun. ACM* **59(3)**, 79

Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R. S., Peng, Y., Reddivari, P., Doshi, V., and Sachs, J.: 2004, Swoogle: A search and metadata engine for the semantic web, in *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, CIKM '04, New York, NY, USA, Association for Computing Machinery, 652–659 p.

Donnelly, K.: 2006, SNOMED-CT: The advanced terminology and coding system for eHealth., *Studies in health technology and informatics* **121**, 279

Doran, P.: 2009, *Ontology modularization : principles and practice*, *Ph.D. thesis*, University of Liverpool, UK

Doran, P., Palmisano, I., and Tamma, V. A. M.: 2008, SOMET: algorithm and tool for SPARQL based ontology module extraction, in *Proceedings of the Workshop on Ontologies: Reasoning and Modularity, WoMO 2008, Tenerife, Spain, June 2, 2008*

Doran, P., Tamma, V. A. M., and Iannone, L.: 2007, Ontology module extraction for ontology reuse: an ontology engineering perspective, in *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, CIKM 2007, Lisbon, Portugal, November 6-10, 2007*, 61–70 pp

ESWC2016: 2016, *Top-K Shortest Path in Large Typed RDF Graphs Challengemon*, http://2016.eswc-conferences.org/top-k-shortest-path-large-typed-rdf-graphs-challenge.html, Last accessed on 26.01.2020

Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D. L., and Patel-Schneider, P. F.: 2001, Oil: an ontology infrastructure for the semantic web, *IEEE Intelligent Systems* **16(2)**, 38

Fernández-López, M., Gómez-Pérez, A., and Juristo, N.: 1997, Methontology: from ontological art towards ontological engineering, in *Proc. Symposium on Ontological Engineering of AAAI*

Freitas, A., Schmidt, D., Panisson, A., Meneguzzi, F., Vieira, R., and Bordini, R. H.: 2014, Semantic representations of agent plans and planning problem domains, in F. Dalpiaz, J. Dix, and M. B. van Riemsdijk (eds.), *Engineering Multi-Agent Systems*, Cham, Springer International Publishing, 351–366 pp

Gannon, D., Barga, R., and Sundaresan, N.: 2017, Cloud-native applications, *IEEE Cloud Computing* **4(5)**, 16

García-Santa, N., Atemezing, G. A., and Villazón-Terrazas, B.: 2015, The ProtégéLOV Plugin: Ontology Access and Reuse for Everyone, in F. Gandon, C. Guéret, S. Villata, J. Breslin, C. Faron-Zucker, and A. Zimmermann (eds.), *The Semantic Web: ESWC 2015 Satellite Events*, Springer International Publishing, 41–45 pp

Genesereth, M.: 1998, *Knowledge Interchange Format*, Proposed Draft NCITS.T2/98-004, American National Standard

Ghazvinian, A., Noy, N. F., and Musen, M. A.: 2011, How orthogonal are the OBO Foundry ontologies?, *Journal of biomedical semantics* **2 Suppl 2**, S2

Gheorghe, R., Hinman, M. L., and Russo, R.: 2015, *Elasticsearch in Action*, Manning

Giray, G. and Ünalır, M. O.: 2013, A method for ontology-based semantic relatedness measurement, *Turkish Journal of Electrical Engineering and Computer Science* **21**, 420

Gomez-Perez, A. and Corcho, O.: 2001, Ontology languages for the semantic web, *IEEE Intelligent Systems* **17(1)**, 54

Gruber, T. R.: 1993, A translation approach to portable ontology specifications, *Knowledge Acquisition* **5(2)**, 199

Grüninger, M. and Fox, M.: 1995, Methodology for the Design and Evaluation of Ontologies, in *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing, April 13, 1995*

Guarino, N.: 1998, Formal ontology and information systems, in N. Guarino (ed.), *Proceedings of Formal Ontology in Information System*, IOS Press, 3-15 pp

Guizzardi, G.: 2005, *Ontological foundations for structural conceptual models*, *Ph.D. thesis*, University of Twente

Halper, M., Ochs, C., Perl, Y., Arabandi, S., and Musen, M. A.: 2018, Quality assurance of ontology content reuse., in P. Jaiswal, L. Cooper, M. A. Haendel, and C. J. Mungall (eds.), *ICBO*, Vol. 2285 of *CEUR Workshop Proceedings*, CEUR-WS.org

Hanna, J., Cheng, C., Crow, A., Hall, R. A., Liu, J., Pendurthi, T., Schmidt, T., Jennings, S. F., Brochhausen, M., and Hogan, W. R.: 2012, Simplifying mireot; a MIREOT protege plugin, in *Proceedings of the ISWC 2012 Posters & Demonstrations Track, Boston, USA, November 11-15, 2012*

Harris, S. and Seaborne, A.: 2013, *SPARQL 1.1 Query Language, Property Paths*, https://www.w3.org/TR/sparql11-query/#propertypaths, Last accessed on 28.12.2019

Herre, H.: 2010, *General Formal Ontology (GFO): A Foundational Ontology for Conceptual Modelling*, 297–345 pp

Hirst, G. and St-Onge, D.: 1998, Lexical chains as representations of context for the detection and correction of malapropisms, in C. Fellbaum (ed.), *WordNet: An Electronic Lexical Database*, MIT Press, 305–332 pp

Horrocks, I.: 2002, DAML+OIL: A Reason-able Web Ontology Language, in *Proceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '02, Berlin, Heidelberg, Springer-Verlag, 2–13 pp

Hulpuş, I., Prangnawarat, N., and Hayes, C.: 2015, Path-Based Semantic Relatedness on Linked Data and Its Use to Word and Entity Disambiguation, in *Proceedings of the 14th International Conference on The Semantic Web - ISWC 2015 - Volume 9366*, Berlin, Heidelberg, Springer-Verlag, 442–457 pp

Hussain, S. and Abidi, S.: 2009, A Rule-based Method for Extracting RDF (S) and OWL Sub-ontologies, in *Canadian Semantic Web Working Symposium. Semantic Web and Beyond, Springer (May 24 2009)*, Springer

Jahid, M. J. and Ruan, J.: 2012, A Steiner tree-based method for biomarker discovery and classification in breast cancer metastasis, *BMC Genomics* **13(Suppl 6)**, S8

Jonquet, C., Musen, M. A., and Shah, N. H.: 2010, Building a biomedical ontology recommender web service, *Journal of biomedical semantics* **1 Suppl 1(Suppl 1)**, S1

Kalibatiene, D. and Vasilecas, O.: 2011, Survey on ontology languages, in J. Grabis and M. Kirikova (eds.), *Perspectives in Business Informatics Research*, Berlin, Heidelberg, Springer Berlin Heidelberg, 124–141 pp

Kamdar, M. R., Tudorache, T., and Musen, M. A.: 2017, A systematic analysis of term reuse and term overlap across biomedical ontologies., *Semantic Web* **8(6)**, 853

Katsumi, M. and Grüninger, M.: 2016, What Is Ontology Reuse?, in R. Ferrario and W. Kuhn (eds.), *Formal Ontology in Information Systems - Proceedings of the 9th International Conference, {FOIS} 2016, Annecy, France, July 6-9, 2016*, Vol. 283 of *Frontiers in Artificial Intelligence and Applications*, {IOS} Press, 9–22 pp

Katsumi, M. and Grüninger, M.: 2017, Choosing ontologies for reuse, *Applied Ontology* **12(3-4)**, 195

Keet, C. M. and Ławrynowicz, A.: 2016, Test-Driven Development of Ontologies, in H. Sack, E. Blomqvist, M. D'Aquin, C. Ghidini, S. P. Ponzetto, and C. Lange (eds.), *The Semantic Web. Latest Advances and New Domains*, Cham, Springer International Publishing, 642–657 pp

Kendall, E. F. and McGuinness, D. L.: 2019, *Ontology Engineering*, Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool Publishers

Lin, D.: 1998, An Information-Theoretic Definition of Similarity, in *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 296–304 pp

Martínez-Romero, M., Jonquet, C., O'Connor, M. J., Graybeal, J., Pazos, A., and Musen, M. A.: 2017, NCBO Ontology Recommender 2.0: an enhanced approach for biomedical ontology recommendation, *Journal of Biomedical Semantics* **8(1)**, 21

Martínez-Romero, M., Vázquez-Naya, J. M., Pereira, J., and Pazos, A.: 2014, Bioss: A system for biomedical ontology selection, *Computer Methods and Programs in Biomedicine* **114(1)**, 125

Mazuel, L. and Sabouret, N.: 2008, Semantic relatedness measure using object properties in an ontology, in A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, and K. Thirunarayan (eds.), *The Semantic Web - ISWC 2008*, Berlin, Heidelberg, Springer Berlin Heidelberg, 681–694 pp

McBride, B.: 2002, Jena: a semantic web toolkit, *IEEE Internet Computing* **6(6)**, 55

McGuinness, D. L.: 2003, Ontologies come of age, in *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential [outcome of a Dagstuhl seminar]*, 171–194 pp

Mcguinness, D. L., Fikes, R., Hendler, J., and Stein, L. A.: 2002, Daml+oil: an ontology language for the semantic web, *IEEE Intelligent Systems* **17(5)**, 72

Miao, Z., Wang, J., Zhou, B., Zhang, Y., and Lu, J.: 2008, Method for extracting rdf(s) sub-ontology, in *2008 International Conference on Cyberworlds*, 348-353 pp

Morris, J. and Hirst, G.: 2004, Non-classical lexical semantic relations, in *Proceedings of the HLT-NAACL Workshop on Computational Lexical Semantics*, CLS '04, Stroudsburg, PA, USA, Association for Computational Linguistics, 46–51 pp

Musen, M. A.: 2015, The Protégé Project: A Look Back and a Look Forward., *AI matters* **1(4)**, 4

Nair, J. and Tudorache, T.: 2011, *Bioportal Import Plugin*, https://protegewiki.stanford.edu/wiki/BioPortal_Import_Plugin, Last accessed on 14.08.2019

Nair, J., Tudorache, T., Whetzel, T., Noy, N. F., and Musen, M. A.: 2011, The bioportal import plugin for protégé, in *Proceedings of the 2nd International Conference on Biomedical Ontology, Buffalo, NY, USA, July 26-30, 2011*

Noy, N. F. and McGuinness, D. L.: 2001, *Ontology Development 101: A Guide to Creating Your First Ontology*, Technical report

Noy, N. F. and Musen, M. A.: 2004, Specifying Ontology Views by Traversal, in S. A. McIlraith, D. Plexousakis, and F. van Harmelen (eds.), *The Semantic Web – ISWC 2004*, Berlin, Heidelberg, Springer Berlin Heidelberg, 713–725 pp

Ochs, C., Perl, Y., Geller, J., Arabandi, S., Tudorache, T., and Musen, M. A.: 2017, An empirical analysis of ontology reuse in bioportal, *Journal of Biomedical Informatics* **71**, 165

Øhrstrøm, P., Andersen, J., and Schärfe, H.: 2005, What has happened to ontology, in F. Dau, M.-L. Mugnier, and G. Stumme (eds.), *Conceptual Structures: Common Semantics for Sharing Knowledge*, Berlin, Heidelberg, Springer Berlin Heidelberg, 425–438 pp

Peralta, D., Pinto, H. S., and Mamede, N.: 2003, Reusing a Time Ontology, 121–128 pp

Peroni, S.: 2017, A simplified agile methodology for ontology development, in M. Dragoni, M. Poveda-Villalón, and E. Jimenez-Ruiz (eds.), *OWL: Experiences and Directions – Reasoner Evaluation*, Cham, Springer International Publishing, 55–69 pp

Pinto, H. S. and Martins, J. P.: 2000, Reusing Ontologies, *Proceedings of the*

AAAI 2000 Spring Symposium on Bringing Knowledge to Business Processes 77–84 pp

Pinto, H. S., Staab, S., Tempich, C., and Sure, Y.: 2006, *Distributed Engineering of Ontologies (DILIGENT)*, 303–322 pp, Springer Berlin Heidelberg, Berlin, Heidelberg

Pirró, G.: 2009, A semantic similarity metric combining features and intrinsic information content, *Data and Knowledge Engineering* **68(11)**, 1289

Pulido, J. R. G., Ruiz, M. A. G., Herrera, R., Cabello, E., Legrand, S., and Elliman, D.: 2006, Ontology languages for the semantic web: A never completely updated review, *Knowledge-Based Systems* **19(7)**, 489

Ranwez, V., Ranwez, S., and Janaqi, S.: 2012, Subontology extraction using hyponym and hypernym closure on is-a directed acyclic graphs, *IEEE Transactions on Knowledge and Data Engineering* **24(12)**, 2288

Rector, A. L., Rogers, J. E., Zanstra, P. E., and Van Der Haring, E.: 2003, OpenGALEN: open source medical terminology and tools., *AMIA ... Annual Symposium proceedings. AMIA Symposium* **2003**, 982

Resnik, P.: 1995, Using Information Content to Evaluate Semantic Similarity in a Taxonomy, in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 448–453 pp

Rosse, C. and Mejino, J. L. V.: 2003, A reference ontology for biomedical informatics: the Foundational Model of Anatomy, *Journal of Biomedical Informatics* **36(6)**, 478

Russ, T., Valente, A., Macgregor, R., and Swartout, W.: 1999, Practical Experiences in Trading Off Ontology Usability and Reusability

Sabou, M., Dzbor, M., Baldassarre, C., Angeletou, S., and Motta, E.: 2007, Watson: A gateway for the semantic web, in *Poster session of the European Semantic Web Conference, ESWC*

Sadalage, P. J. and Fowler, M.: 2013, *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*, Addison-Wesley, Upper Saddle River, NJ

Sadeghi, A. and Fröhlich, H.: 2013, Steiner tree methods for optimal sub-network identification: An empirical study, *BMC Bioinformatics* 14(1)

Salvadores, M., Alexander, P. R., Musen, M. A., and Noy, N. F.: 2013, BioPortal as a Dataset of Linked Biomedical Ontologies and Terminologies in RDF., *Semantic web* **4(3)**, 277

Seidenberg, J.: 2009, *Web Ontology Segmentation: Extraction, Transformation, Evaluation*, 211–243 pp, Springer Berlin Heidelberg, Berlin, Heidelberg

Simperl, E.: 2009, Reusing ontologies on the semantic web: A feasibility study, *Data Knowl. Eng.* **68(10)**, 905

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y.: 2007, Pellet: A practical owl-dl reasoner, *Journal of Web Semantics* **5(2)**, 51 , Software Engineering and the Semantic Web

Smith, B., Kumar, A., and Bittner, T.: 2005, *Basic Formal Ontology for Bioinformatics*, IFOMIS Reports

Sánchez, D. and Batet, M.: 2013, A semantic similarity method based on information content exploiting multiple ontologies, *Expert Systems with Applications* **40(4)**, 1393

SpringBoot: 2020, *Spring Boot*, https://spring.io/projects/spring-boot, Last accessed on 26.01.2020

Stardog: 2020, *Stardog*, https://www.stardog.com/, Last accessed on 26.01.2020

Studer, R., Benjamins, V., and Fensel, D.: 1998, Knowledge engineering: Principles and methods, *Data & Knowledge Engineering* **25(1-2)**, 161

Suárez-Figueroa, M. C., García-Castro, R., Villazón-Terrazas, B., and Gómez-Pérez, A.: 2011, Essentials in ontology engineering: Methodologies, languages, and tools, in *Proceedings of the 2nd Workshop organized by the eeb data models community- CIB conference*, 9–21 pp

Suárez-Figueroa, M. C., Gómez-Pérez, A., and Fernández-López, M.: 2012, *The NeOn Methodology for Ontology Engineering*, 9–34 pp, Springer Berlin Heidelberg, Berlin, Heidelberg

Suchanek, F. M., Kasneci, G., and Weikum, G.: 2008, Yago: A large ontology from wikipedia and wordnet, *Journal of Web Semantics* **6(3)**, 203 , World Wide Web Conference 2007Semantic Web Track

Sun, Y., Ma, C., and Halgamuge, S.: 2017, The node-weighted Steiner tree approach to identify elements of cancer-related signaling pathways, *BMC Bioinformatics* 18(February 2018)

Takahashi, H. and Matsuyama, a.: 1980, An approximate solution for the Steiner problem in graphs, *Math Japonica* **24(6)**, 573

Tversky, A.: 1977, Features of similarity, *Psychological Review* **84(4)**, 327

Uschold, M. and Grüninger, M.: 1996, Ontologies: principles, methods, and applications, *Knowledge Engineering Review* **11(2)**, 93

Uschold, M. and Gruninger, M.: 2004, Ontologies and semantics for seamless connectivity, *ACM SIGMOD Record* **33(4)**, 58

Uschold, M., Healy, M., Williamson, K., Clark, P., and Woods, S.: 1998, Ontology reuse and application, in *PROCEEDINGS OF THE 1ST INTERNATIONAL CONFERENCE ON FORMAL ONTOLOGY IN INFORMATION SYSTEMS(FOIS'98*, IOS Press, 179–192 pp

Uschold, M. and King, M.: 1995, Towards a methodology for building ontologies, in *Workshop on Basic Ontological Issues in Knowledge Sharing (IJCAI)*

van Harmelen Deborah L. McGuinness, F.: 2004, *OWL Web Ontology Language Overview*, Technical report

Vandenbussche, P.-Y., Atemezing, G., Poveda-Villalón, M., and Vatant, B.: 2017, Linked Open Vocabularies (LOV): A gateway to reusable semantic vocabularies on the Web, *Semantic Web Journal* **8**, 437

Virtuoso: 2020, *Virtuoso*, https://virtuoso.openlinksw.com/, Last accessed on 26.01.2020

Voß, S.: 1992, Steiner's problem in graphs: heuristic methods, *Discrete Applied Mathematics* **40(1)**, 45

Vrandečiundefined, D.: 2012, Wikidata: A new platform for collaborative data collection, in *Proceedings of the 21st International Conference on World Wide Web*, New York, NY, USA, Association for Computing Machinery, 1063–1064 p.

Vukotic, A., Watt, N., Abedrabbo, T., Fox, D., and Partner, J.: 2014, *Neo4j in Action*, Manning

Wächter, T. and Schroeder, M.: 2010, Semi-automated ontology generation within OBO-Edit., *Bioinformatics (Oxford, England)* **26(12)**, i88

Whetzel, P. L., Noy, N., Shah, N., Alexander, P., Dorf, M., Fergerson, R., Storey, M. A., Smith, B., Chute, C., and Musen, M.: 2011, BioPortal: Ontologies and integrated data resources at the click of a mouse, *CEUR Workshop Proceedings*

833(April), 292

Wilkinson, M. and et al: 2016, The fair guiding principles for scientific data management and stewardship, *Nature Scientific Data* (160018)

Xiang, Z., Courtot, M., Brinkman, R. R., Ruttenberg, A., and He, Y.: 2010, OntoFox: web-based support for ontology reuse., *BMC research notes* **3**, 175

Zhang, Y., Vasconcelos, W., and Sleeman, D.: 2005, Ontosearch: An ontology search engine, in M. Bramer, F. Coenen, and T. Allen (eds.), *Research and Development in Intelligent Systems XXI*, London, Springer London, 58–69 pp

ACKNOWLEDGEMENT

Firstly, I’d like to express my sincere gratitude to my thesis adviser Assoc. Prof. Dr. Murat Osman ÜNALIR for his all support, patience, motivation, and knowledge. His guidance, experience, and knowledge have always helped me, not only for this thesis study and Ph.D education, but also for all my academic and professional life. Besides, I would like to thank to all thesis monitoring committee members, Assoc. Prof. Dr. Murat KOMESLİ and Asst. Prof. Dr. Özgür GÜMÜŞ, for their valuable comments and contributions.

I would also like to thank to Dr. Görkem GİRAY, for his help and guidance. His semantic relatedness studies have been an important resource for both my master and doctorate thesis.

The last but not the least, I would like to give my special thanks to all my family, my wife Mümine, my daughter Öykü, my son Deniz, and lastly my mom Gülperi, for their help and patience.

CURRICULUM VITAE

Cemil ABİŞ

Date of Birth	:	16.08.1985
Place of Birth	:	İzmir
Nationality	:	T.C.
Email	:	cemilabis@gmail.com
Linkedin	:	https://www.linkedin.com/in/cemilabis
Github	:	https://github.com/cemilabis

EXPERIENCE

01/2020 - Present	:	Logo Software, İzmir Software Development Consultant
01/2017 - 12/2019	:	İzmir Katip Çelebi University, İzmir Software Team Leader
05/2014 - 12/2016	:	EES, İzmir Software Team Leader
04/2011 - 05/2014	:	Netsis Software, İzmir, Technology Development Team Leader
08/2007 - 03/2011	:	Birim Information Technologies, İzmir, Software Specialist

EDUCATION

Master **(2008-2011)**	:	Ege University Department of Computer Engineering
Bachelor **(2003-2007)**	:	Ege University Department of Computer Engineering

HONORS AND AWARDS

Highest Ranked Student in Computer Engineering Department, 2003-2007

Second Highest Ranked Student in Engineering Faculty, 2003-2007

FIELDS OF INTEREST

Ontologies, linked data, ontology reuse, ontology modularization, sub-ontology extraction

APPENDICES

APPENDIX-1: TECHNOLOGY STACK

This section includes technologies, such as programming languages, databases, IDEs and other tools, used in developmet of the thesis study.

Programming Languages

Name	Usage Purpose	Version
Java	Development of ontology reuse server and client	1.8
Python	Development of statistics and analysis tools	3.6
Javascript	Development of web client	EcmaScript 6

Databases

Name	Usage Purpose	Version
Neo4j	Storing RDF graph data	3.2.3
MongoDb	Storing statistics	4.0
ElasticSearch	Storing gull text data to enabled search operations	2.4.0

IDEs

Name	Usage Purpose	Version
Intellij Idea	Development of all Java related software	2016.3.4
Netbeans	Development and design of Java forms	11
Visual Studio Code	Development of Python and HTML based applications	1.33.1
PyCharm	Development of Python applications	2018.2.3

Other Tools and Libraries

Name	Usage Purpose	Version
OWL API	Development of OWL related operations	5.1.0
Spring Framework	Development of core application and REST endpoints	4.3.0
Protege	API development	5.5.0
NodeJS	Development of Web applications	12.14.1

APPENDIX-2: SOURCE CODES

Name of application	Programming Language	Github Address
Ontorogist (Ontology modularization and reuse application)	Java	https://github.com/cemilabis/ontorogist
Protege Plugin	Java	-
Web Application	Html, Javascript	https://github.com/cemilabis/ontorogist-web
Performance evaluation	Python	https://github.com/cemilabis/ontorogist-tests
Statistics	Python	https://github.com/cemilabis/ontorogist-statistics
ETL	Java	-
Steiner Tree Comparison	R	-

*Source codes of all applications developed under the thesis (including those on the github) are delivered with CD.

APPENDIX-3 : DATASET-1

This data set has been created manually. In each sub-dataset, concepts which are thought to be more related to each other are selected. Thus, it is aimed to measure the reuse more accurately.

Namespace abbreviations

- cpr: http://purl.org/cpr/
- obo: http://purl.obolibrary.org/obo/
- bfo: http://www.ifomis.org/bfo/1.1/span#

```
<obo:OGMS_0000014>
<cpr:screening-act>

<cpr:clinical-diagnosis>
<cpr:clinician-role>

<obo:OGMS_0000029>
<obo:OGMS_0000085>

<cpr:sign-recording >
<cpr:clinical-act>

<cpr:sign-recording >
<cpr:screening-act>

<obo:OGMS_0000060>
<cpr:clinical-examination>

<obo:OGMS_0000029>
<obo:OGMS_0000085>
<obo:OGMS_0000028>

<obo:OGMS_0000014>
<cpr:screening-act>
<cpr:clinical-artifact>

<cpr:diagnostic-procedure>
<obo:OGMS_0000029>
```

```
<cpr:clinical-artifact>

<obo:OGMS_0000045>
<obo:OGMS_0000029>
<obo:OGMS_0000020>

<obo:OGMS_0000045>
<cpr:clinical-act>
<obo:OGMS_0000031>

<cpr:patient-role>
<cpr:clinician-role>
<bfo:Process>

<obo:OGMS_0000045>
<obo:OGMS_0000029>
<obo:OGMS_0000020>
<obo:OGMS_0000031>

<obo:OGMS_0000045>
<cpr:clinical-act>
<obo:OGMS_0000031>
<cpr:patient>

<cpr:patient-role>
<cpr:clinician-role>
<bfo:Process>
<cpr:laboratory-test>

<cpr:diagnostic-procedure>
<obo:OGMS_0000029>
<cpr:clinical-artifact>
<cpr:physician-role>

<cpr:diagnostic-procedure>
<obo:OGMS_0000029>
<cpr:clinical-artifact>
<cpr:physician-role>
<cpr:laboratory-test-finding>

<cpr:patient>
<cpr:patient-role>
<cpr:clinician-role>
<bfo:Process>
```

```
<cpr:laboratory-test>

<obo:OGMS_0000045>
<obo:OGMS_0000029>
<obo:OGMS_0000020>
<obo:OGMS_0000031>
<obo:OGMS_0000073>

<obo:OGMS_0000045>
<cpr:clinical-act>
<obo:OGMS_0000031>
<cpr:patient>
<cpr:clinical-artifact>

<obo:OGMS_0000045>
<cpr:clinical-act>
<obo:OGMS_0000031>
<cpr:patient>
<cpr:clinical-artifact>
<cpr:medical-device>

<cpr:patient>
<cpr:patient-role>
<cpr:clinician-role>
<bfo:Process>
<cpr:laboratory-test>
<obo:OGMS_0000015>

<obo:OGMS_0000045>
<cpr:clinical-act>
<obo:OGMS_0000031>
<cpr:patient>
<cpr:clinical-artifact>
<cpr:screening-act>
```

APPENDIX-4 : DATASET-2

This data set has been created randomly and is only used for performance evaluation. The elements of each subset in this dataset has been selected based on the reuse weight of the ontology in SDO. If an ontology is reused more frequently in SDO, the elements of this ontology is included more frequently in Dataset-2.

Namespace abbreviations

- cpr: http://purl.org/cpr/
- obo: http://purl.obolibrary.org/obo/
- bfo: http://www.ifomis.org/bfo/1.1/span#
- fma: http://purl.org/sig/ont/fma/

```
<obo:OGMS_0000080>
<obo:OGMS_0000028>

<fma:fma49908>
<obo:IAO_0000409>

<fma:fma6447>
<cpr:immaterial-anatomical-continuant>

<obo:OGMS_0000089>
<bfo:ProcessualContext>

<fma:fma7310>
<fma:fma68630>

<bfo:ProcessBoundary>
<fma:fma7184>

<cpr:SurgicalMethod>
<obo:OGMS_0000027>

<fma:fma10703>
<bfo:ProcessAggregate>
```

```
<fma:fma68624>
<obo:OGMS_0000069>

<bfo:ProcessualEntity>
<cpr:anatomical-space>

<obo:IAO_0000078>
<fma:fma9576>
<cpr:sign-recording>

<cpr:laboratory-test-finding>
<cpr:anatomical-space>
<bfo:ScatteredSpatiotemporalRegion>

<fma:fma61823>
<fma:fma4452>
<fma:fma6434>

<cpr:immaterial-pathological-continuant>
<obo:IAO_0000409>
<fma:fma61993>

<cpr:patient-role>
<cpr:anatomical-structure>
<fma:fma61284>

<obo:OGMS_0000093>
<cpr:pathogen>
<fma:fma61817>

<obo:OGMS_0000075>
<fma:fma6314>
<fma:fma20358>

<obo:OGMS_0000069>
<fma:fma68627>
<cpr:infectious-disease>

<cpr:medical-device>
<fma:fma12289>
<cpr:procedure>

<bfo:ProcessBoundary>
<fma:fma73950>
```

```
<fma:fma6313>

<cpr:physical-therapy>
<bfo:TemporalInstant>
<fma:fma62792>
<fma:fma49893>

<fma:fma6418>
<bfo:SpatiotemporalInstant>
<fma:fma4544>
<fma:fma51893>

<bfo:TemporalInstant>
<cpr:anamnesis>
<obo:OGMS_0000054>
<cpr:state>

<cpr:physician>
<bfo:ScatteredTemporalRegion>
<obo:OGMS_0000048>
<fma:fma10429>

<cpr:pathological-role>
<cpr:longitudinal-patient-medical-history>
<obo:IAO_0000225>
<obo:OGMS_0000033>

<cpr:clinical-phenotype>
<obo:IAO_0000078>
<fma:fma71019>
<bfo:ProcessualContext>

<fma:fma7162>
<bfo:ScatteredSpatiotemporalRegion>
<obo:OGMS_0000093>
<fma:fma4245>

<fma:fma72719>
<obo:IAO_0000225>
<fma:fma73995>
<bfo:ProcessAggregate>

<cpr:extra-organismal-continuant>
<cpr:immaterial-anatomical-continuant>
```

```
<cpr:clinical-diagnosis>
<cpr:diagnostic-image>

<obo:OGMS_0000064>
<obo:IAO_0000409>
<fma:fma62955>
<obo:OGMS_0000040>

<bfo:Process>
<obo:OGMS_0000018>
<bfo:SpatiotemporalRegion>
<fma:fma61859>

<fma:fma68616>
<cpr:pharmacological-substance>
<fma:fma86254>
<obo:OGMS_0000074>
<cpr:patient>

<bfo:SpatiotemporalInstant>
<cpr:material-pathological-entity>
<bfo:ProcessAggregate>
<fma:fma71064>
<obo:OGMS_0000073>

<obo:OGMS_0000015>
<cpr:physical-therapy>
<obo:OGMS_0000059>
<cpr:screening-act>
<cpr:sign-recording>

<cpr:state>
<obo:OGMS_0000075>
<obo:IAO_0000225>
<cpr:pathogen>
<bfo:TemporalInstant>

<fma:fma50720>
<bfo:TemporalInterval>
<obo:IAO_0000078>
<cpr:procedure>
<obo:OGMS_0000051>

<obo:OGMS_0000069>
```

```
<obo:IAO_0000409>
<cpr:immaterial-anatomical-continuant>
<fma:fma71066>
<fma:fma10703>

<fma:fma5865>
<fma:fma67687>
<fma:fma58616>
<obo:OGMS_0000055>
<obo:OGMS_0000085>

<cpr:self-examination>
<fma:fma83017>
<fma:fma73906>
<bfo:ScatteredSpatiotemporalRegion>
<cpr:medication>

<fma:fma6435>
<cpr:patient-record>
<cpr:pharmacological-substance>
<cpr:screening-act>
<obo:OGMS_0000056>

<obo:OGMS_0000088>
<obo:IAO_0000409>
<bfo:ProcessBoundary>
<fma:fma45626>
<fma:fma6167>
<bfo:TemporalInterval>

<obo:OGMS_0000091>
<fma:fma71062>
<cpr:morphologic-alteration>
<fma:fma51244>
<obo:OGMS_0000027>
<obo:OGMS_0000079>

<fma:fma7154>
<obo:OGMS_0000056>
<cpr:recorded-clinical-situation>
<fma:fma83465>
<obo:IAO_0000078>
<fma:fma14754>
```

```
<fma:fma71023>
<fma:fma54439>
<obo:OGMS_0000082>
<obo:IAO_0000225>
<bfo:FiatProcessPart>
<obo:IAO_0000078>

<obo:IAO_0000078>
<bfo:ScatteredTemporalRegion>
<cpr:anatomical-surface>
<cpr:anatomical-space>
<obo:OGMS_0000083>
<fma:fma73962>

<fma:fma50866>
<obo:IAO_0000409>
<fma:fma58274>
<bfo:ProcessualContext>
<cpr:patient-record>
<obo:OGMS_0000085>

<obo:OGMS_0000081>
<bfo:ProcessualContext>
<obo:OGMS_0000057>
<obo:OGMS_0000066>
<cpr:laboratory-test>
<bfo:ProcessualEntity>

<fma:fma66644>
<obo:OGMS_0000091>
<obo:OGMS_0000057>
<fma:fma6417>
<bfo:SpatiotemporalInterval>
<obo:OGMS_0000090>

<fma:fma14900>
<bfo:SpatiotemporalInterval>
<fma:fma68629>
<bfo:SpatiotemporalInstant>
<cpr:clinical-administration-act>
<obo:OGMS_0000049>

<bfo:SpatiotemporalInstant>
<fma:fma55662>
```

```
<fma:fma5022>
<obo:OGMS_0000055>
<cpr:self-examination>
<fma:fma71027>

<fma:fma14755>
<cpr:immaterial-pathological-continuant>
<obo:OGMS_0000065>
<fma:fma71209>
<fma:fma86196>
<fma:fma73992>
<cpr:anatomical-surface>

<fma:fma14750>
<cpr:medication>
<fma:fma71040>
<cpr:laboratory-test-finding>
<fma:fma6305>
<fma:fma25072>
<fma:fma14749>

<cpr:vital-sign>
<cpr:medical-problem>
<bfo:ScatteredSpatiotemporalRegion>
<bfo:ProcessualEntity>
<cpr:physical-therapy>
<obo:OGMS_0000075>
<bfo:SpatiotemporalInterval>

<fma:fma6444>
<cpr:sequela>
<cpr:immaterial-pathological-continuant>
<fma:fma6415>
<obo:OGMS_0000090>
<obo:OGMS_0000084>
<fma:fma51895>

<fma:fma7088>
<cpr:anatomical-structure>
<cpr:physician>
<obo:IAO_0000409>
<cpr:anatomical-space>
<obo:OGMS_0000056>
<fma:fma22930>
```

```
<fma:fma73921>
<cpr:physical-anatomical-entity>
<obo:IAO_0000409>
<obo:OGMS_0000079>
<fma:fma51244>
<fma:fma9905>
<cpr:deoxyribonucleic-acid-structure>

<fma:fma86187>
<obo:IAO_0000078>
<obo:OGMS_0000084>
<obo:IAO_0000409>
<fma:fma72249>
<obo:OGMS_0000088 bfo:Process>

<fma:fma12224>
<fma:fma14743>
<cpr:immaterial-pathological-continuant>
<fma:fma7161>
<fma:fma14745>
<fma:fma14764>
<cpr:healthcare-professional-role>

<fma:fma80184>
<obo:OGMS_0000085>
<cpr:clinical-examination>
<cpr:laboratory-test>
<obo:OGMS_0000081>
<cpr:pathological-disposition>
<cpr:anamnesis>

<fma:fma54448>
<fma:fma24879>
<cpr:morphologic-alteration>
<obo:OGMS_0000090>
<obo:OGMS_0000086>
<fma:fma4058>
<obo:IAO_0000409>
<obo:OGMS_0000054>

<cpr:immaterial-pathological-continuant>
<obo:OGMS_0000056>
<obo:OGMS_0000026>
```

```
<cpr:recorded-clinical-situation>
<fma:fma14763>
<fma:fma6446>
<fma:fma68605>
<obo:OGMS_0000078>

<cpr:anatomical-structure>
<fma:fma20357>
<fma:fma4567>
<obo:OGMS_0000057>
<fma:fma49945>
<bfo:ProcessualContext>
<obo:OGMS_0000026>
<bfo:Process>

<obo:OGMS_0000080>
<bfo:ProcessAggregate>
<cpr:patient>
<cpr:laboratory-test>
<bfo:ProcessualContext>
<cpr:syndrome>
<bfo:ScatteredSpatiotemporalRegion>
<cpr:anatomical-structure>

<fma:fma83465>
<cpr:substance-administration>
<cpr:medication>
<cpr:sign-recording>
<cpr:physician-role>
<fma:fma71056>
<cpr:patient-record>
<obo:OGMS_0000025>

<fma:fma4147>
<obo:OGMS_0000091>
<obo:OGMS_0000013>
<fma:fma14749>
<fma:fma68624>
<obo:OGMS_0000056>
<fma:fma45623>
<obo:OGMS_0000084>

<obo:OGMS_0000070>
<bfo:TemporalInstant>
```

```
<cpr:physician-role>
<cpr:immaterial-pathological-continuant>
<cpr:pathogen>
<obo:IAO_0000225>
<bfo:Process>
<cpr:diagnostic-procedure>

<fma:fma14746>
<fma:fma61830>
<fma:fma37319>
<fma:fma61823>
<bfo:Process>
<obo:IAO_0000409>
<obo:OGMS_0000074>
<bfo:ScatteredTemporalRegion>

<fma:fma6410>
<fma:fma71067>
<cpr:laboratory-test-finding>
<bfo:SpatiotemporalRegion>
<bfo:TemporalInterval>
<obo:OGMS_0000074>
<fma:fma71021>
<fma:fma14740>

<fma:fma61905>
<obo:OGMS_0000084>
<fma:fma73934>
<obo:OGMS_0000057>
<fma:fma6305>
<fma:fma6298>
<fma:fma68623>
<fma:fma83865>

<fma:fma6414>
<fma:fma83153>
<bfo:ProcessualContext>
<obo:OGMS_0000068>
<cpr:diagnostic-procedure>
<obo:IAO_0000225>
<bfo:SpatiotemporalInterval>
<fma:fma50737>
<fma:fma22706>
```

```
<cpr:medical-problem>
<obo:OGMS_0000078>
<bfo:TemporalInstant>
<obo:OGMS_0000068>
<fma:fma83465>
<bfo:TemporalInstant>
<fma:fma6427>
<cpr:image>
<bfo:ScatteredSpatiotemporalRegion>

<obo:OGMS_0000046>
<fma:fma45638>
<fma:fma12294>
<cpr:clinical-administration-act>
<bfo:TemporalInstant>
<fma:fma4245>
<obo:OGMS_0000085>
<bfo:ScatteredTemporalRegion>
<obo:IAO_0000078>

<fma:fma71040>
<obo:OGMS_0000037>
<cpr:patient>
<fma:fma10703>
<fma:fma62003>
<fma:fma50720>
<obo:OGMS_0000066>
<cpr:syndrome>
<obo:OGMS_0000073>
```

APPENDIX-5 : API Contracts

```
SearchResponse
{
  "ontologyResults": [
  {
    "items": [ {
      "id": "string",
      "label": "string",
      "ontology": "string"
    }],
    "ontologyName": "string"
  }],
  "result": true,
  "resultMesssage": "string"
}

GetClassResponse
{
  "item": {
    "displayName": "string",
    "id": "string",
    "ontologyName": "string"
  },
  "result": true,
  "resultMesssage": "string"
}

GetClassesResponse
{
  "items": [ {
    "displayName": "string",
    "id": "string",
    "ontologyName": "string"
  }],
  "result": true,
  "resultMesssage": "string"
}

GetClassDefinitionResponse
{
  "items": [ {
    "object": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    },
    "predicate": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    },
    "subject": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    }
  }],
  "result": true,
  "resultMesssage": "string"
}

GetClassRelationResponse
{
  "items": [ {
    "object": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    },
    "predicate": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    },
    "subject": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    }
  }],
  "result": true,
  "resultMesssage": "string"
}

FindPathRequest
{
  "from": {
    "displayName": "string",
    "id": "string",
    "ontologyName": "string"
  },
  "to": {
    "displayName": "string",
    "id": "string",
    "ontologyName": "string"
  },
}

FindPathResponse
{
  "path": [ {
    "object": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    },
    "predicate": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    },
    "subject": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    }
  }],
  "result": true,
  "resultMesssage": "string"
}

FindPathsResponse
{
  "paths": [
  [
    {
      "object": {
        "displayName": "string",
        "id": "string",
        "ontologyName": "string"
      },
      "predicate": {
        "displayName": "string",
        "id": "string",
        "ontologyName": "string"
      },
      "subject": {
        "displayName": "string",
        "id": "string",
        "ontologyName": "string"
      }
    }]
  ],
  "result": true,
  "resultMesssage": "string"
}

FindSubOntologyRequest
{
  "classes": [ {
    "displayName": "string",
    "id": "string",
    "ontologyName": "string"
  } ],
  "parameters": {
    "includeDomainAndRange": true,
    "includeEquivalentClasses": true,
    "includeProperties": true,
    "includeSuperClasses": true
  },
  "triples": [ {
    "object": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    },
    "predicate": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    },
    "subject": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    }
  }]
}

FindSubOntologyResponse
{
  "result": true,
  "resultMesssage": "string",
  "triples": [ {
    "object": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    },
    "predicate": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    },
    "subject": {
      "displayName": "string",
      "id": "string",
      "ontologyName": "string"
    }
  } ]
}

DownloadSubOntologyResponse
{
  "result": true,
  "resultMesssage": "string",
  "subOntologyUrl": "string"
}
```

APPENDIX-6: GRAPHML SAMPLE USED IN STEINER-TREE PERFORMANCE EVALUATION

This data set contains five ontologies which are reused by SDO (see Section 7.2).These ontologies used in Steiner Tree performance data set contains 2332 vertices and 6298 edge. A partial graphml of these ontologies is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
↪  xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
↪  http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="directed">
    <node id="n1298" labels=":Bioportal:Biotop">
      <data key="labels">:Bioportal:Biotop</data>
      <data key="resourceId">_:genbotid277</data>
      <data key="url">_:genbotid277</data>
    </node>
    <node id="n1299" labels=":Bioportal:Biotop">
      <data key="labels">:Bioportal:Biotop</data>
      <data key="resourceId">_:genbotid278</data>
      <data key="url">_:genbotid278</data>
    </node>
    <node id="n1300" labels=":Bioportal:Biotop">
      <data key="labels">:Bioportal:Biotop</data>
      <data key="resourceId">_:genbotid281</data>
      <data key="url">_:genbotid281</data>
     </node>
    <node id="n1301" labels=":Bioportal:Biotop">
    <data key="labels">:Bioportal:Biotop</data>
    <data key="resourceId">_:genbotid279</data>
    <data key="url">_:genbotid279</data>
  </node>
  <node id="n1302" labels=":Bioportal:Biotop">
    <data key="labels">:Bioportal:Biotop</data>
    <data key="resourceId">_:genbotid280</data>
    <data key="url">_:genbotid280</data>
  </node>
  <node id="n1303" labels=":Bioportal:Biotop">
    <data key="labels">:Bioportal:Biotop</data>
    <data key="resourceId">biotop:NoncanonicalStaticProcessualEntity</data>
    <data key="url">biotop:NoncanonicalStaticProcessualEntity</data>
  </node>
  <node id="n1304" labels=":Bioportal:Biotop">
    <data key="labels">:Bioportal:Biotop</data>
    <data key="resourceId">biotop:CarbohydrateMoleculeOrResidue</data>
    <data key="url">biotop:CarbohydrateMoleculeOrResidue</data>
  </node>
  <node id="n1305" labels=":Bioportal:Biotop">
    <data key="labels">:Bioportal:Biotop</data>
    <data key="resourceId">biotop:CarbohydrateMonomer</data>
    <data key="url">biotop:CarbohydrateMonomer</data>
  </node>
  <node id="n1306" labels=":Bioportal:Biotop">
    <data key="labels">:Bioportal:Biotop</data>
    <data key="resourceId">_:genbotid283</data>
```

```
      <data key="url">_:genbotid283</data>
    </node>
    <node id="n1307" labels=":Bioportal:Biotop">
      <data key="labels">:Bioportal:Biotop</data>
      <data key="resourceId">biotop:CarbohydrateSequenceInformation</data>
      <data key="url">biotop:CarbohydrateSequenceInformation</data>
    </node>
    <node id="n1308" labels=":Bioportal:Biotop">
      <data key="labels">:Bioportal:Biotop</data>
      <data key="resourceId">biotop:CatalyticRole</data>
      <data key="url">biotop:CatalyticRole</data>
    </node>
    <node id="n1309" labels=":Bioportal:Biotop">
      <data key="labels">:Bioportal:Biotop</data>
      <data key="resourceId">biotop:ChemicalRole</data>
      <data key="url">biotop:ChemicalRole</data>
    </node>
    <node id="n1310" labels=":Bioportal:Biotop">
      <data key="labels">:Bioportal:Biotop</data>
      <data key="resourceId">biotop:CategorizationSystem</data>
      <data key="url">biotop:CategorizationSystem</data>
    </node>
    <node id="n1311" labels=":Bioportal:Biotop">
      <data key="labels">:Bioportal:Biotop</data>
      <data key="resourceId">biotop:IntellectualProduct</data>
      <data key="url">biotop:IntellectualProduct</data>
    </node>
```

```
<node id="n1312" labels=":Bioportal:Biotop">
  <data key="labels">:Bioportal:Biotop</data>
  <data key="resourceId">biotop:RegulationOrLaw</data>
  <data key="url">biotop:RegulationOrLaw</data>
</node>
<node id="n1313" labels=":Bioportal:Biotop">
  <data key="labels">:Bioportal:Biotop</data>
  <data key="resourceId">biotop:Causing</data>
  <data key="url">biotop:Causing</data>
</node>
<node id="n1314" labels=":Bioportal:Biotop">
  <data key="labels">:Bioportal:Biotop</data>
  <data key="resourceId">biotop:CellInVivo</data>
  <data key="url">biotop:CellInVivo</data>
</node>
<node id="n1315" labels=":Bioportal:Biotop">
  <data key="labels">:Bioportal:Biotop</data>
  <data key="resourceId">_:genbotid284</data>
  <data key="url">_:genbotid284</data>
</node>
...........
...........
...........
<edge id="e597" source="n537" target="n538" label="obo:hasDbXref">
  <data key="label">obo:hasDbXref</data>
  <data key="url">obo:hasDbXref</data>
</edge>
```

```
    <edge id="e598" source="n539" target="n155" label="owl:inverseOf">
      <data key="label">owl:inverseOf</data>
      <data key="url">owl:inverseOf</data>
    </edge>
    <edge id="e599" source="n156" target="n111" label="rdf:type">
      <data key="label">rdf:type</data>
      <data key="url">rdf:type</data>
    </edge>
    <edge id="e600" source="n156" target="n540" label="owl:equivalentProperty">
      <data key="label">owl:equivalentProperty</data>
      <data key="url">owl:equivalentProperty</data>
    </edge>
    <edge id="e601" source="n540" target="n541" label="owl:inverseOf">
      <data key="label">owl:inverseOf</data>
      <data key="url">owl:inverseOf</data>
    </edge>
    <edge id="e602" source="n541" target="n542" label="owl:equivalentProperty">
      <data key="label">owl:equivalentProperty</data>
      <data key="url">owl:equivalentProperty</data>
    </edge>
    <edge id="e603" source="n542" target="n156" label="owl:inverseOf">
      <data key="label">owl:inverseOf</data>
      <data key="url">owl:inverseOf</data>
    </edge>
    <edge id="e604" source="n541" target="n156" label="owl:inverseOf">
      <data key="label">owl:inverseOf</data>
      <data key="url">owl:inverseOf</data>
    </edge>
    <edge id="e605" source="n156" target="n153" label="rdfs:subPropertyOf">
      <data key="label">rdfs:subPropertyOf</data>
      <data key="url">rdfs:subPropertyOf</data>
    </edge>
    <edge id="e606" source="n156" target="n543" label="rdfs:subPropertyOf">
      <data key="label">rdfs:subPropertyOf</data>
      <data key="url">rdfs:subPropertyOf</data>
    </edge>
    <edge id="e607" source="n543" target="n154" label="owl:inverseOf">
      <data key="label">owl:inverseOf</data>
      <data key="url">owl:inverseOf</data>
    </edge>
    <edge id="e608" source="n156" target="n544" label="owl:inverseOf">
      <data key="label">owl:inverseOf</data>
      <data key="url">owl:inverseOf</data>
    </edge>
    <edge id="e609" source="n544" target="n156" label="owl:inverseOf">
      <data key="label">owl:inverseOf</data>
      <data key="url">owl:inverseOf</data>
    </edge>
    <edge id="e610" source="n156" target="n419" label="rdf:type">
      <data key="label">rdf:type</data>
      <data key="url">rdf:type</data>
    </edge>
    <edge id="e611" source="n521" target="n111" label="rdf:type">
      <data key="label">rdf:type</data>
      <data key="url">rdf:type</data>
    </edge>
    ...........
    ...........
    ...........
  </graph>
</graphml>
```